SIMULTANEOUS PLANNING AND CONTROL FOR AUTONOMOUS GROUND VEHICLES

By

THOMAS C. GALLUZZO

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2006

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2006		**00-00-2006 to 00-00-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Simultaneous Planning and Control for Autonomous Ground Vehicles	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Center for Intelligent Machines and Robotics,Department of Mechanical Engineering,University of Florida,Gainesville,FL,32611**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**157**	

I dedicate this dissertation entirely to my mother and father, Janice and Dennis Galluzzo. Their complete and unwavering support of my education has made this achievement possible.

ACKNOWLEDGMENTS

I would first like to thank Dr. Carl Crane for his kind support throughout my graduate education. He has been an excellent advisor for both my academic and field research. I would also like to thank my committee, Dr. Warren Dixon (co-chair), Dr. B.J. Fregly, Dr. John Schueller, and Dr. Antonio Arroyo, for their support and guidance.

This work has been made possible by the Air Force Research Laboratory at Tyndall Air Force Base, Florida. Thanks go to Al Neese, Dr. Jeff Wit and the rest of their staff. Also, I would like to thank the Air Force project manager at CIMAR, David Armstrong. He has been a good friend and mentor throughout this work.

Finally I would like to thank my close colleagues and friends at CIMAR, namely Danny Kent, Bob Touchton, Sanjay Solanki, Roberto Montane, and Chad Tobler. They have made this journey all the more exciting and worthwhile.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

SIMULTANEOUS PLANNING AND CONTROL FOR AUTONOMOUS GROUND VEHICLES

By

Thomas C Galluzzo

December 2006

Chair: Carl Crane
Cochair: Warren Dixon
Major Department: Mechanical and Aerospace Engineering

Motion planning and control for autonomous vehicles are complex tasks that must be done in order for a ground robot to operate in a cluttered environment. This dissertation presents the theory, implementation, and test results for some new and novel Receding Horizon Control (RHC) techniques that allow these tasks to be unified into one.

The first new method is called Heuristic Receding Horizon Control (HRHC), and uses a modified A* search to fulfill the online optimization required by RHC. The second is called Dual-Frequency Receding Horizon Control (DFRHC), and is used to simplify the trajectory planning process during the RHC optimization.

Both methods are combined together to form a practical implementation, which is discussed in detail. The autonomous ground vehicle, the NaviGator, developed at the Center for Intelligent Machines and Robotics, serves as a platform for the implementation and testing discussed.

Finally, data and their analysis are presented. The results obtained help to support the theoretical and practical claims made by this dissertation.

CHAPTER 1
INTRODUCTION

Everyday more and more robotic vehicles are entering the real world. They are being put to work just about everywhere manual vehicles have been used in the past. From agriculture, and mining operations, to inside factories and hospitals, they are increasing safety, efficiency, and performance in all tasks otherwise considered to be too dull, dirty or dangerous for manual labor.

Unmanned vehicles are being used intensely by militaries worldwide. The United States for example, has already been using unmanned air vehicles for several years to patrol the skies over conflicts in foreign lands. Recently, by congressional mandate, the U.S. Army has set a goal to have one-third of all operational ground combat vehicles operating unmanned by the year 2015 [US00]. This is a difficult feat that, if achieved, will save countless American lives on the battlefields of tomorrow.

The recent explosion of unmanned vehicle technology has been made possible by vast improvements in sensors, computers and research developments. There is now a greater understanding of the problems that need to be solved in order to allow autonomous machines to operate in the largely uncertain real world. Yet despite all of the advancements, there is still room for improvement and much work to be done.

Background

Unmanned vehicles are designed to perform a variety of tasks, which they perform with varying levels of independence. While some unmanned machines are rigidly controlled by human operators, via telemetry and wireless input, others are sent about with little to no assistance. These are the type of unmanned vehicles under consideration in this dissertation. They fall into the category known as autonomous vehicles. Autonomous vehicles are operated under human control at the highest of levels. Instructions here may simply command the vehicle

to reach a goal point or follow a corridor. Commands may also be issued on an even higher level describing an abstract mission, such as patrolling a perimeter, or sweeping through a defined area. At these levels, the robot is given a higher amount of command and control authority. Consequently, the less input provided by the operator, the more influence the machine has over its own actions.

Autonomous vehicles pose a number of unique problems in their design and implementation. There is no longer a human-in-the-loop control scheme for the vehicle. The unmanned system itself must close the loop from environment feedback to low-level vehicle control. Where a human operator would normally analyze data feedback from telemetry, remote video, etc. and then decide the best course of action, designers must now instrument the vehicle so it can automate these tasks. This requires the inclusion of internal state and environmental sensors, along with onboard computers and software capable of processing the sensed information and planning the vehicle's action accordingly. One way of formalizing this overall process is known as the sense-plan-act paradigm for robotic development [NIL98]. It is a breakdown of the complete design into compartmentalized tasks and processes, which allows for ease of implementation of the whole system in general.

The first design step in the sense-plan-act paradigm is the inclusion of different types of sensors onto the vehicle platform. These sensors serve two general purposes. The first is to measure the state of the vehicle itself, such as its position, orientation, speed, and perhaps also health monitoring information such as temperatures, pressures, etc. In humans, this is known as proprioception, a word derived from the combination of the Latin proprius, meaning "one's own" and perception. It is a vital part of the robotic system; without proprioceptive sensors the vehicle

would not have the feedback necessary to be able to control itself, regardless of environmental conditions.

The complement of proprioception is exteroception. This is the system's ability to sense information originating outside of itself. It is the ability to sense one's environment. Sensors such as cameras and range detectors provide this information. The job of the system designer is to outfit the autonomous vehicle with those sensors necessary and appropriate to provide the correct environment feedback, thus allowing the system to decide how to act within it.

A key note of importance is that accurate internal state estimates are critical in order to be able to make sense out of exteroceptive information. An example that helps to understand this is the estimation of a camera's orientation on a vehicle. Without knowing a camera's orientation in the environment, it is impossible for the robot to be able to know where the sensed images are coming from. This means that the robot must be aware of its own orientation before it can use the camera information. The same is true for other environment sensors, and thus it is necessary to have valid proprioceptive state estimates before analyzing exteroceptive information.

Designers face the problem of ensuring the validity of information from both types of sensors. This problem becomes very difficult in the presence of noise and other uncertainty, which is always the case in real world implementations, and therefore it requires careful attention from design through implementation.

The second step in the sense-plan-act design is giving the autonomous vehicle the ability to calculate how to react to sensed internal and external information. This step requires the unmanned vehicle to have the necessary processing and computational power along with the algorithms and software capable of providing robust and stable control laws that guide the navigation of the robot. This step replaces the decision making and input provided by an

operator, such as with teleoperated control. The decision making process overall produces the desired response based upon the mission objective of the autonomous vehicle.

Action is the final step in the paradigm. At this phase, all of the sensed data have been processed and analyzed, and the autonomous vehicle commands its own inputs. As with all unmanned vehicles, input commands are delivered to the actuators that allow the vehicle to produce motion: engine fuel valves, amplified electric motors, brakes, and many others. Autonomous vehicles generate their own decisions at the planning level. These govern how to drive the vehicle actuators, which cause the platform to move. The sense-plan-act sequence continues on repeatedly, allowing the vehicle to self-regulate.

This paradigm and its steps described can be applied to all autonomous vehicles, and in fact all autonomous robots. However, it is specifically used in this dissertation for the design and application of autonomous ground vehicles (AGVs), although other types of vehicles may benefit from the topic proposed.

There are many shapes and sizes of AGVs. Different methods of propulsion for AGVs have been explored by a number of researchers. There are skid-steered and differential drive vehicles which translate and turn by means of two sets of independent wheels or tracks on either side of the vehicle platform. There are also car-like vehicles, which move by rotating a set of wheels, and turn by deflecting the relative angle between the wheels and the vehicle chassis. Many combinations of propulsion and turning exist in car-like vehicles: front, rear, and all-wheel drive, for example, are propulsion methods commonly used among them.

There are several unique problems facing AGV engineers that are not of concern for other types of unmanned vehicles. The machine environment poses the greatest problem for a successful AGV. Unlike air and water unmanned vehicles, which can operate in a vast

uncluttered space, AGVs must often operate within confined spaces, among static and dynamic obstacles, and on different qualities of terrain. Avoiding collisions with obstacles and refraining from becoming trapped is a hard challenge to overcome. The vehicle must be able to quickly and accurately realize its environment, so designers must incorporate robust sensors capable of resolving the complexity of the surroundings. The vehicle must also have a high degree of mobility with the ability to respond quickly to avoid potential collisions. Finally, the robot must be equipped with enough computational power to be able to quickly process the large amounts of sensor data, and then control its response safely.

Focus

The method by which a ground robot can plan and control its own motion is the subject of this research. AGV motion planning and control are difficult problems for many reasons. First, they require the analysis of multidimensional data from multiple sensors. This means that control algorithms must be able to handle a relatively high throughput of data, and be fast enough (on the machines that perform them) to maintain vehicle stability and performance. Second, the data must be processed with consideration for any uncertainties in sensing and vehicle response.

As aforementioned, uncertainty is always a concern when machines are operating in the real world. These uncertainties can be attributed to several sources, some examples include sensor limitations, noise and the inherent unpredictability of operating environments. Uncertainty in vehicle response is attributable to the fact that machines can only respond to their inputs with a limited degree of repeatability. External disturbances and wear are examples of variation sources that affect how a vehicle will respond to a given input. By minding these influences during the data processing and planning phase, an AGV is far more likely to respond correctly in its situation.

Another problem to motion planning and control is that there must be consideration for the motion constraints of any actuators involved or the vehicle platform itself. This is especially an important issue for car-like vehicles because they are subject to a nonholonomic constraint. This means that although a vehicle driving on a surface may have three degrees of freedom, (translation in two dimensions and rotation in one) it can only translate in the direction it is immediately facing. Consequently, the equations of motion describing the vehicle dynamics are non-integrable, which makes the problem much harder to solve. This also means that car-like vehicles are under actuated. In other words, the number of control inputs to the system is less than the number of degrees of freedom in the system's configuration space. This is illustrated by the fact that a car-like vehicle can only move by input of a steering wheel and the rotation of its drive wheels, yet given the right control sequence, it can assume any position and orientation. This is the nature of the problem undertaken in this dissertation.

Problem statement

A formal description of the general AGV motion planning and control problem can be formulated as follows:

Given:

Sensed data describing the local environment around the vehicle, and a goal structure (point, line, trajectory, etc.) which the vehicle is desired to reach, or track, and also, feedback estimates of the full vehicle state, i.e. position, velocity, and orientation.

Develop:

An algorithm capable of optimizing and executing the vehicle's motion through its local environment, and obtaining the goal. The algorithm must be able to maintain throughput of sensor data and command the vehicle actuators accordingly.

Motivation

For over a decade The Center for Intelligent Machines and Robotics (CIMAR) has been actively pursuing research in the field of autonomous robotics and AGV technology. A key challenge during this endeavor has been tackling the problem of motion planning and control.

The research of AGVs at CIMAR has primarily been conducted under the direct support of the Air Force Research Laboratory (AFRL) at Tyndall Air Force Base. The technology developed at CIMAR under this program has advanced the fields of automated surveying and mapping, unexploded ordinance detection, mine field clearing, and modular system architecture design.

Over the years, many different groups of people at CIMAR have successfully automated over ten unmanned ground vehicles. The latest of theses vehicles is called the NaviGator, shown in Figure 1-1. It is a front-wheel steered, all-wheel drive platform.

The NaviGator is a custom built all-terrain vehicle, with a mild steel roll bar frame. It has 9" Currie axles, Bilstein Shocks, hydraulic steering, and front and rear disk brakes with rear emergency brakes. It has a Honda 150 HP transverse engine mounted longitudinally. A locked transaxle connected to the engine drives front and rear differentials. It has two integrated 28 volt alternators that generate 4800 watts of continuous electrical power. This power is delivered to onboard computers, actuators, and other electronics, along with a ¾ Ton air conditioning unit that cools an enclosure which houses most of the vehicle's electrical equipment. The NaviGator is equipped with a number of sensors caged on the front of the vehicle. The sensors include a radar unit, cameras, and three scanning laser range detectors. All are used for environment sensing. The vehicle also has an integrated GPS/INS system, which is used for estimating its position, orientation and speed.

This vehicle was used by CIMAR as an entry to the DARPA Grand Challenge national competition. DARPA is the Defense Advanced Research Projects Agency, a small division of the United States Department of Defense, and in 2004 through 2005 it sponsored the Grand Challenge competition in an effort to rapidly advance experience and innovation in AGV technology. The competition was designed in a manner that would allow participating research groups to help accelerate national research in this field, without diverting resources from other ongoing government projects.

The goal of the competition was to build and field a robotic vehicle that could traverse over 150 miles of desert terrain without any human control. This was a technological feat that, prior to the 2005 competition, had never been accomplished. However, in October 2005, five teams entered vehicles that successfully completed the entire challenge course.

Although team CIMAR's NaviGator was not able to finish the competition, it did advance as one of 23 Finalists (out of over 200 applicants) and complete over 15 miles of the course, which demonstrated the tremendous effort put forth by all team members. The NaviGator is also considered a success because it will continue to serve as a research and development platform at CIMAR for years to come.

The development of an online path planning and control algorithm for the NaviGator has been the driving motivational force behind this research topic. As part of the effort to enter the NaviGator into the 2005 DARPA Grand Challenge, a new approach to motion planning and control was developed. The approach is a variation on the receding horizon control strategy, where a sequence of open-loop commands are repeatedly optimized and delivered to the system. In a typical receding horizon controller, the optimal set of commands is calculated by minimizing a quadratic cost formula. The technique devised for the NaviGator is unique in that it

obtains a near optimal solution via a heuristic search. This has several advantages and disadvantages that will be discussed in detail in the following chapters. The fundamental significance is that this technique provided a well fit motion planning and control solution for the NaviGator. Although many other techniques and approaches exist, the research and advancement of this technique may benefit other implementations where it is suited.

Figure 1-1. Picture of the NaviGator AGV in race configuration, which was taken just before the DARPA Grand Challenge in October, 2006.

CHAPTER 2
REVIEW OF THE LITURATURE

To compare and contrast the newly devised control strategy, a review of published research literature has been conducted. Various researchers have explored different methodologies for the AGV motion planning and control problem. The majority of the methods brake down the problem into two individual tasks.

In one task, sensor or a priori data are analyzed and a geometric path or a time varying trajectory of the robot's motion is planned through an environment or workspace. These motion describing structures are often optimized for travel time, distance, or minimum collision potential. Sometimes combinations of parameters are optimized. The way in which the planning problem is formulated also varies greatly between applications. As input to the planning algorithms researchers use different techniques to describe the local environment. While some use discrete raster images or vector geometry, others use continuous mathematical functions. The different formulations have unique characteristics that will be described in detail in this chapter.

The control task attempts to solve the problem of regulating the vehicle in order to execute a previously determined motion command. This command can be as simple as a desired position and orientation, or as complex as a trajectory sequence requiring specific turning maneuvers and speed changes. As with planning, the techniques developed for the control task are diverse. Many researchers have struggled with and found viable solutions for dealing with the nonholonomic and other constraints of AGVs. Although the methods differ greatly in implementation, there is as always, a tradeoff between stability, robustness, and performance.

The review of research has been broken down into exploring the input and motion structures and then the planning and control algorithms themselves. Input structures represent sensor and other required data delivered to the planning or control algorithms. Likewise the

results of the planning algorithms are delivered to controllers via a motion structure. By understanding the different input and output structures, a greater understanding of the algorithms and techniques is gained.

Planning and control input structures

Potential Fields

A category of input structures exist in the form of mathematically described functions and fields. One of the earliest types of fields explored is known as a potential field. In 1985 Khatib [KHA85] presented an obstacle avoidance approach for manipulators and mobile robots based on the "Artificial Potential Field" concept, where obstacles were represented as potential energy fields that generated repulsive forces, and goal configurations for the robot were represented as attractive gravitational fields. The resultant imaginary force acting on the robot was used to guide its motion.

Since then researchers have used potential fields in countless scenarios. Barraquand et al. generate collision free paths for a three degree of freedom mobile robot using potential fields [BAR92]. Their approach to path planning consists of incrementally building a graph connecting the local minima of a potential function defined over the configuration space of the mobile robot. A search was conducted over the graph until a goal configuration was obtained. This was an efficient approach that did not require any pre-computation steps (as other researchers had suggested) to be performed over the potential field function.

Warren addresses the issue of global path planning with potential fields [WAR89]. Here planning of a manipulator and mobile robot motion was done in configuration space (C-space), a multi-dimensional space described by a set of generalized coordinates which represent the position and orientation of a rigid body. In his implementation, Warren constructs an arbitrary trial path in the configuration space of the robot, which connects the initial configuration to the

goal configuration via a set of straight line segments. The entire path is then modified under the influence of the potential field until a minimum potential path is found. The effect of modifying the entire path at once greatly reduced the problem of becoming trapped in a local minimum of the potential field, however, a collision-free path could still not be guaranteed.

Several limitations of potential field methods applied to AGVs were identified by Koren and Borenstein [KOR91]. Specifically they identified four problems that could arise when a mobile robot was subjected to the imaginary forces of obstacles and a goal. First, a mobile robot could become trapped in the local minima of a potential field; as could occur when the goal configuration was blocked by a U-shaped obstacle. (This phenomenon of potential field methods was previously identified by a number of other researchers; see Andrews et al. [AND83] and Tilove [TIL90].) Second, robots could often favor longer paths to travel around any closely spaced obstacles that they encountered, rather than simply pass untouched between them. Third, the robot motion could become unstable when passing nearby obstacles, and finally, oscillations could occur when a robot is traveling down a narrow corridor. This is because repulsive forces of walls close to either side of the robot caused oscillatory motion when it was subjected to a small disturbance. For these and other reasons, researchers continued to pursue other ways of formulating the planning and control problem.

Navigation Functions

A special kind of potential field function, known as a navigation function, was introduced in 1988 by Rimon and Koditschek [RIM88]. The navigation function is unique in that the only minima occurs at the goal configuration for the robot, no other local minima exists. This prevented the robot from becoming stuck in any local minima that might exist in a traditional potential field. The navigation function techniques however, are susceptible to other drawbacks. For a proposed application offered by Rimon [RIM91] the robot was required to operate in a

circular space inhabited only by disk shaped obstacles. This is a scenario with little practicality in a real-world environment, and only considered effective in a highly controlled experiment. Another limitation is that the navigation function can be difficult or expensive to compute.

Other researchers have attempted to overcome these limitations. Konkimalla and LaValle were able to compute navigation functions for an arbitrary shaped workspace containing arbitrary clutter [KON99]. The navigation functions were computed numerically for nonholonomic systems, which they demonstrated by using their techniques to generate optimal motions for car-like robots in simulation. With their method, the navigation function was computed over a uniformly distributed quantized free configuration space. This was described by a set of points in the robot's n-dimensional configuration, which excluded the space occupied by any obstacles. However, since the robot was not constrained to occupy only discrete points in the space, the navigation function was interpolated (with a novel technique) between points in order to allow the vehicle to maintain smooth trajectories, and also to keep the free configuration space continuous. This method was also unique in that it computed the navigation function iteratively over a propagating wave-front, which originated at the goal configuration. This was a key to allowing arbitrary shapes in describing the navigation function. A drawback of the methodology developed is that it assumed the robot was operating in a static and predetermined environment. Thus, the navigation function algorithm developed by Konkimalla and LaValle, was still not suitable for real-time application.

One real-time strategy for using navigation functions to control a robot operating in a dynamic environment, was proposed by Loizou et al. in 2003 [LOI03]. Their approach was to develop a non-smooth navigation function, which allowed for faster computation than a smooth function. To further simplify computation, the obstacles in the dynamic workspace were assumed

to be disk shaped, as with Rimon's application. The approach was proven to work in simulation, and guaranteed collision avoidance and motion convergence. However, arbitrarily shaped obstacles could not be accounted for.

Velocity Fields

Another type of field used to control an AGV is known as a velocity field. Here the environment of the robot is assumed to have an imaginary flow field. The direction and magnitude of the field at any given point describes the desired velocity of the robot. This concept was pioneered by a number of researchers in the early 1990's. Li and Horowitz first published a paper on the subject in 1993 [LI93]. In their research they remarked that velocity fields had an advantage over a traditional potential, or navigation functions, in that they accounted for the robot's desired motion over its complete workspace. In other methods, the path that a robot followed in order to reach its goal could not be predetermined without integrating the dynamics functions. Using velocity fields to describe the desired motion removed that ambiguity, because the robot's desired speed and orientation is specified at all possible configurations.

Li and Horowitz extended their initial work by applying the velocity field concept specifically to robot contour following problems [LI96]. In this research, a velocity field was constructed in a manner that would result in the robot tracing out a desired geometric contour. This was made possible because the resulting control scheme applied to the robot ensured convergence onto a stable limit cycle, which was equivalent to the desired contour.

A novel methodology for calculating desired velocity fields was suggested by Keymeulen and Decuyper [KEY94], in which a field could be generated via fluid dynamics. In what they describe as a metaphorical approach, by placing an imaginary fluid source at the robot, a fluid sink at its destination, and constraining the boundary conditions of the workspace and obstacles, the streamlines of the resulting fluid flow could be used to describe the desired path of the robot.

They showed that this was a very powerful approach because it was not susceptible to local minima, and also the imaginary flow would be able to instantly adapt to any dynamic topology. The major drawback of the fluid dynamics approach is the very expensive computation necessary when recalculating the flow field upon any change in the robot's environment. At the time of publication, the authors suggested that it was well suited on a parallel, analog, or optical computing machine. However, as computing machinery continues to advance, this powerful method becomes more and more applicable to real-world implementations.

Dixon et al. were able to establish a control scheme that allowed a nonholonomic Wheeled Mobile Robot (WMR) to track a desired velocity field [DIX05]. This extended the work other researchers had done, which did not account for nonholonomic systems. The group developed an adaptive controller, and employed a Lyapunov-based analysis to prove global asymptotic tracking of the velocity field.

Occupancy Grids

Mobile robots are often designed to operate in environments that are unstructured and unknown. In these cases, the system has no a priori model or knowledge of its surroundings. Occupancy grid structures offer a means for a robot to map and rationalize this unknown space. An occupancy grid is a multidimensional tessellation (or array) of space into uniformly shaped cells. Each cell in the array contains a probability estimate that identifies whether the space is occupied or empty.

The earliest uses of occupancy grids for mobile robot planning and control were developed by Alberto Elfes at Carnegie Mellon University. Elfes initially described the implementation of occupancy grids on mobile robots in a series of papers published in the mid 1980's. Initially they were used for a sonar-based mapping and navigation system [ELF86]. Here the general issues pertaining to occupancy grids, such as spatial resolution and sensor uncertainties were described

conceptually. Several years later, the formalized mathematics and problem structures were presented [ELF89]. In this research, derived estimates of the cells were obtained by interpreting range readings using probabilistic sensor models. A Bayesian estimation procedure was employed to accurately calculate evolving grid cell states as sensor readings repeated in time.

Borenstein and Koren presented an approach that combined the concepts of occupancy grids and potential fields [BOR89]. In their method known as the Virtual Force Field, each occupied cell in the grid applied an imaginary repulsive for to the robot. The magnitude of the force was proportional to the probability that the cell is truly occupied by an obstacle. This method yielded promising results, in its robustness to sensor uncertainty, however it still was susceptible to the drawbacks of potential fields described in the previous section.

A classification of occupancy grids, described by Elfes as inference grids (where cells contain an estimate of multiple possible states) was employed by Touchton et al. at CIMAR [TOU06]. In their implementation, a type of structure named traversibiliy grid stores an estimate of the quality of terrain contained in the robot's local environment. Here each cell expresses the degree to which a robot would be capable of successfully passing through the contained space. This level of classification was felt necessary in order to accommodate AGV navigation off-road, where space can rarely be considered only occupied or empty.

Geometric Models

In some of the earliest path planning research, the robot's environment and workspace were represented with geometric primitives such as lines, polygons and circles. These structures were used because they required only a minimal amount of computational memory (a resource more limited at the time than today).

Research conducted by Lozano-Perez and Wesley [LOZ79], involved the planning of a polyhedral object moving among other known polyhedral objects. In this original work, a graph

known as a visibility graph was formulated to find free paths for the moving object. The graph was constructed by connecting straight line segments between the vertices of the polyhedrons. It was called a visibility graph, because connected vertices could "see" each other unobstructed from the polyhedral objects.

A different approach to planning among polygonal obstacles was presented by Takahashi and Schilling [TAK89]. In their method, the free space of the robot's environment was represented with a generalized Voronoi diagram, which is the locus of points equidistant from two of more obstacle boundaries including the workspace boundary. For a polygonal workspace inhabited by polygonal objects, the diagram consists of only linear and parabolic line segments. The Voronoi diagram method made for more efficient planning, in that it consists of fewer vertices than a visibility graph, and also has the advantage of keeping the motion plan further away from obstacles. However, the method in general produces larger travel distances, which can reduce performance.

A method for maintaining minimum distance optimality, and increasing computation efficiency was proposed by Rohnert [ROH87]. In this method, a structure called a tangent graph was constructed, where common tangent lines connect convex polygons in the robot's workspace. It is more efficient because it contains fewer vertices than a visibility graph. A means of computing the tangent graph was suggested by Liu and Arimoto [LIU94]. The research group proposed an algorithm called the moving-line algorithm, which efficiently computed the graphs by decomposing the construction task into two sub-problems: detecting common tangents, and intersection checking among the tangents and obstacles. Their algorithm performance was demonstrated with a number of simulated experiments and the results were presented.

Motion Command Structures

The simplest input command structure to regulate a mobile robot's motion is a position and orientation set-point, in which the robot is desired to maintain a fixed and specified pose. Another commonly used motion structure represents the path geometry of the mobile robot together with an associated speed profile. Complete time varying trajectories are also common in practice. In this case, the vehicle's complete position and orientation is denoted as a function of time.

A broad overview of the different motion commands has been detailed by De Luca et al. [DEL98] in a chapter of the book: Robot Motion Planning and Control, edited by Jean-Paul Laumond. In their discussion they point out that these three motion structures can sometimes be cast as a higher level problem or sub-problem of one another. For example, some research has suggested that the path regulation problem is a subset of the higher level trajectory tracking problem.

In their discussion, they analyze the controllability of a car-like robot attempting these three tasks. Their analysis employs a useful tool known as the Lie Algebra Rank Condition (see [ISI95]), which allows the controllability of a drift-less nonlinear system to be tested. Along with this test they exemplify that set-point stabilization of a car-like robot cannot be achieved via a smooth time-invariant feedback control law. A result established on the basis of Brockett's theorem [BRO83], which implies that a necessary condition for smooth stabilization of a system is that the number of inputs equals the number of states. Since this is not the case, such condition is violated.

Despite the varying complexities and difficulties of all three control tasks, feedback control solutions have been identified for them. De Luca and group, present a number of the techniques,

and the associated mathematics for them. Simulated results are also presented in their overview. Some of the strategies are presented later in this chapter.

Planning Strategies and Algorithms

As previously stated, AGV motion planning and control is often divided into a planning task, which generates a motion command for the robot, and a control task, which regulates the vehicle onto the predetermined motion structure. A vast number of methods have been developed for the planning task. They are described here in detail.

Deterministic Geometric Planners

Early research in nonholonomic path planning considered only simple cases where environmental constraints and obstacle avoidance were not part of the problem. Pioneering work was done by Dubins during the late 1950's. He proved that optimal paths connecting a car-like vehicle from an initial position and orientation, to a final configuration state, were made up of straight line segments and circular arcs, with radius equivalent to that of the vehicle's minimum turning radius [DUB57]. This theory was developed for a vehicle traveling only with forward motion. Reeds and Shepp extended the work of Dubins to include motion for a vehicle traveling both forwards and backwards [REE91].

There are several drawbacks to these planning methods. The most notable is that they do not consider the presence of any obstacles in the workspace of the vehicle. Also, the curves are made up of segments with discontinuous curvature at the connections between them. This means that for a car-like robot to follow the curves exactly, it must come to a stop at the segment junctions in order to change its steering angle.

Search and Heuristic Methods

A group of motion planners rely on heuristic or graph searching techniques. With these methods, a plan is constructed that is based on the result of searching through discrete graph representations of possible robot configurations in its obstacle cluttered environment.

A common search technique for these applications is called A* (A-star), and it was originally developed by Hart, Nilsson, and Raphael [HAR68]. Their research presented the formal basis for showing that the search method is both admissible and computationally optimal. Since its inception, A* has been used in a wide variety of mobile robot planning algorithms.

Kuan et al. use an A* search to find a path to a goal configuration for a mobile robot navigating among convex polygon shaped obstacles [KUA85]. Their method locates critical "channels" and "passage regions" within the free space, which are then decomposed into non-overlapping geometric-shaped primitives. From this representation, the path planning algorithm then finds trajectories inside the channels and passage regions.

Another implementation searches through a multi-resolution grid to find a path. Presented by Kambhampati and Davis [KAM86], a method using a quad-tree hierarchical representation of the workspace is exploited to gain a computational savings in the search. Their technique was more efficient because it did not consider the excess detail in parts of the space that did not substantially affect the planning operation.

A minimum time planning approach for a robot was given by Meystel et al. [MEY86]. In this method, the robot's acceleration, deceleration and turning capabilities were taken into consideration during planning in order to minimize the overall time-to-goal for the calculated trajectory. Their algorithm employed the A* search to optimize the plan.

Thorpe and Matthies offer a path relaxation technique for mobile robot planning [THO85]. Here a grid search is performed to calculate and initial trajectory to the goal. Once an initial

solution is established the path is “relaxed” by allowing the nodes that describe the trajectory to follow the negative gradient of the cost grid. This is done to determine a more optimal final solution, and helps plan a path that is not too close to any obstacles.

Vector Methods

In an effort to increase computational efficiency, reduce algorithm complexity, and to correct some of the problems resulting from field-based control methods, a number of researchers have devised methods for planning using vector geometry.

Borenstein and Koren pioneered the Vector Field Histogram method [BOR91]. In this method, a polar histogram of obstacle densities was constructed around a window centered at the robot’s location. The density values in the histogram were calculated by analyzing how objects in a grid interacted with a set of vectors originating at the robot. Once the histogram was calculated a desired heading vector was determined based on a target heading and the valleys in the histogram. Results showed successful local planning, but a possibility for the robot to become “trapped” in dead-end situations existed.

A modification to the previous method was developed by An and Wang [AN04]. Their method known as Vector Polar Histogram differs slightly form Vector Field Histogram in that the histogram is calculated directly from a polar sensor (rather than a grid), and it is transformed into a binary histogram based on a dynamically changing threshold. This threshold value is calculated in real-time and is based on the robot’s velocity. Overall, the modifications were claimed to offer simpler and equally effective local obstacle avoidance.

Vector based planning methods have also been used to provide a simple means for avoiding moving obstacles. A technique involving relative velocity vectors was offered by Fiorini and Shiller [FIO93]. A collision free path was planned for a circular object moving among circular obstacles with constant linear motions. To plan the path a graph was constructed

with straight line segments, which was a generalization of the visibility graph to the case of moving obstacles.

Jing and Wang advance a vector technique for avoiding obstacles in an uncertain environment, specifically for a mobile robot [JIN05]. The dynamic motion planning problem was transformed into an optimization problem in the robot's acceleration space. As with Fiorini and Shiller's method, relative velocity vectors between the robot and encountered obstacles are used in order to define the robot's desired behavior. With the behavior determined, a feedback control law was established, and stability in each planning period was proven.

Probabilistic Planning

Probabilistic methods for path planning have become increasingly popular for mobile robot navigation. These techniques are often designed to guide the robot through free space in a manner that reduces the risk of collisions or other undesired behaviors. Usually, an iterative analysis of the local environment is conducted as a preliminary step to probabilistic planning.

Kavraki et al. developed an algorithm to calculate probabilistic roadmaps of a robot's free configuration space [KAV96]. In what they characterize as the learning phase of the planning method, the roadmaps are constructed by repeatedly generating randomized free configurations of the robot and then connecting these states with a simple motion planner. This process continues until enough configuration nodes exist to successfully plan from a starting configuration to a goal. The number of free configuration nodes generated depends on the intricacy of the configuration space.

A probabilistic method for obstacle avoidance amongst moving obstacles with motion uncertainty is proposed by Miura et al. [MIU99]. In this research, moving obstacles within the robots environment are modeled with motion uncertainty, i.e. the predicted future location of the object is uncertain. The obstacles are also modeled for sensing uncertainty, in which the sensed

instantaneous state of the obstacle is unsure. Based on these probabilistic models, their method repeatedly selects the best motion plan in a decision-theoretic manner, that is, by a one-step look-ahead search in a probabilistic search tree.

Cell decomposition is a path planning method that involves partitioning the free configuration space of the robot into disjoint sets, called cells. Methods to generate the cell sets are often costly due to the complexity of determining whether a cell is entirely contained in the free configuration space or not. Lingelbach presents a probabilistic method to cell decomposition where cells are probabilistically sampled to determine if they are free [LIN04]. This method offered an improvement in the efficiency of cell decomposition for high dimensional configuration spaces.

Thrun et al. present a broad explanation of probabilistic path planning algorithms for robotics [THR05]. Their overview describes a technique known as value iteration, as a solution to a Markov decision process. In this process the state of the robot's environment is assumed to be known, but an allowance for stochastic action effects is maintained. In other words, the robot's response to a given input may be uncertain. The value iteration method in effect produces a probabilistic navigation function, which is used in planning to guide the robot's motion.

Control Strategies and Algorithms

The control task for an AGV involves the regulation of the vehicle onto a predetermined motion command. Here the goal is to minimize any error between the vehicle's state and a desired state. This is done by commanding the vehicle plant inputs in a deliberate manner, which is often specified by a mathematically defined function, or procedure. Many control methods have been developed for this purpose and some are described here in depth.

Kinematics Methods

Research in path tracking control of an AGV has demonstrated successful implementations of kinematic control methodologies, where system equations of the vehicle's motion about a geometric path are used to develop a controller. An early kinematic technique known as pure pursuit was originally developed at Carnegie Mellon University. The solution gained much popularity due to its simplicity and performance.

Coulter, a researcher at Carnegie Mellon, describes an implementation of the pure pursuit path tracking algorithm and some of the stability and performance considerations for it [COU92]. Essentially the method is used to calculate the arc (or curvature) necessary to get the vehicle back on the desired path. This is done by choosing a look-ahead distance, and calculating the goal position on the path at that distance. This leads to the analogy that the vehicle is "pursuing" a moving point, which is always at some distance ahead of itself.

A more in depth analysis of the algorithm was presented by Ollero and Heredia [OLL95]. In their research they presented mathematically formulated stability criteria for a vehicle tracking a straight line and circular arcs. Their work showed that the stability of the closed-loop system was dependant upon the look-ahead distance parameter, and any time delays in feedback data. They also presented simulated results; for varying time delays and tuning on the look-ahead parameter.

Vector pursuit is another kinematic technique devised for path tracking. This method was developed by Wit [WIT00] in his doctoral research at CIMAR, and it involves determining the desired motion of the vehicle based on the theory of screws (introduced by Sir Robert S. Ball in 1900 [BAL00]). In an effort to correct some of the drawbacks of pure pursuit, Wit includes the vehicle's desired orientation at the look-ahead point in the geometric analysis. Vector pursuit allowed for the consideration of both heading and distance errors without any mixed units (which

other methods had) in the mathematical foundation. This resulted in a geometrically meaningful control scheme.

Linear Control Systems

Linear control system theory contains a rich set of analysis and synthesis techniques for designing and implementing controllers. Many of these methods have been developed, tested, and proven for several decades, and are still commonly used in practice today. This is because the methods are often simple and robust in implementation, and because of these reasons, researchers have applied the theories to the control of AGVs.

Nelson and Cox, a team of researchers at AT&T Bell Laboratories, demonstrated the use of a simple proportional control scheme on a mobile robot [NEL88]. The controller was used to guide the vehicle along a set of predetermined path segments. There experimental results showed several problems with the control methodology. For example, the vehicle stability was affected directly by its speed, the faster the motion the less stable the path tracking. Also, transitions between path segments of varying curvature led to initial overshoots, which could not be corrected by control tuning.

In a more recent effort, Choi presents a proportional derivative controller for an autonomous highway vehicle [CHO00]. Here the parameters are designed to be adaptive, in an attempt to correct persistent disturbances from wheel misalignments, unbalanced tire pressure, side wind, etc. The compensator developed provided closed-loop control for the vehicles lateral motion in highway lane following, and was demonstrated in experiment to successfully stabilize the vehicle, and reject disturbances.

A modern robust linear control technique known as H_∞ (H infinity) control, was used by Kim et al. to steer an AGV [KIM01]. This technique was used to synthesize a state-space

controller, which was robust to the quantifiable uncertainty of: system parameters, noise, input signals, etc. In this group's research, the controller developed was tested on a car vehicle tracking a straight road. Initial results showed effective performance in tracking the road, and merited additional experimentation on curved roads.

Nonlinear Control

Researchers in the field of mobile robot control have widely developed nonlinear solutions to the problem. Nonlinear controllers are designed and analyzed with a number of mathematical tools mostly based on the foundation of Lyapunov stability criteria, which will be discussed in chapter three of this document. These methods are used, because the mobile robot control problem is highly nonlinear in nature, and it is not well suited for linearization or other linear control techniques.

Significant results were obtained by Samson during the early 1990's. In this work a smooth time-varying feedback controller was developed to regulate a wheeled mobile robot to an arbitrary set-point [SAM90]. This was a powerful result, because it showed that stable set-point feedback regulation, albeit time-varying, was practical despite the implications of Brockett's condition [BRO83], which proved time-invariant feedback for mobile robot set-point regulation is not possible.

Jiang et al. presented another time-varying feedback result for globally stable tracking of a mobile robot pursuing a reference trajectory that is a function of time [JIA97]. In this work, the group presented simulated results, which validated their theoretical results. The robot kinetics equations were included in the design via an integrator backstepping technique.

Dixon has developed several unified regulation and tracking controllers, which guarantee stability of a wheeled mobile robot [DIX00]. The differentiable kinematic control laws

developed utilize a damped dynamic oscillator in which the frequency of oscillation is tunable. These results led to continued successful work with simulated and experimental results.

Model Predictive Control

Model predictive control is an advanced technique often used in industry to solve optimization problems under certain constraints. A specific form of model predictive control is known as receding horizon control. In this methodology, the control problem is optimized over a finite period of time, and the resulting control signal is applied to the system. This process is continually repeated. In each optimization, the first control signal in the optimized control profile is delivered to the plant. The remaining signals are discarded.

Recently this technique has been applied to mobile robot control. Gu et al. presented a result where a wheeled mobile robot was regulated to an arbitrary set-point using a receding horizon controller [GU05]. Their results showed that stability for the robot could be achieved and simulated data were given. Computation time was the main drawback of their result. They cited that real-time implementation of the controller was not practical given their method of optimization, and suggested further work was necessary in order to find ways of improving the computation efficiency. Binary decision trees and artificial neural networks were two methods suggested for incorporation, by the group.

The following chapter offers the theoretical foundation and analysis for a new and novel methodology used to solve the computational problems faced in the real-time implementation of a receding horizon controller, particularly on an AGV.

CHAPTER 3
THEORETICAL APPROACH

Introduction

Motion planning and control for an AGV are both challenging tasks. A novel methodology for unifying these into one task, while maintaining online computability, is the main contribution of this dissertation. A newly devised heuristically optimized receding horizon controller (HRHC) is proposed for the combined task. The purpose of this chapter is to explain and formalize the mathematical foundation of the approach. Another contribution of this dissertation is a novel extension to the standard receding horizon control scheme. The extension, called Dual-frequency receding horizon control (DFRHC) is also presented in this chapter.

Receding horizon control (RHC), or more generally, model predictive control (MPC) is an advanced technique used to solve complex constrained control problems online. A broad overview of the technique is offered in [MAY00]. It employs methodology from optimal control theory to determine the best control action for a given state. In receding horizon control, a sequence of open-loop plant input commands are optimized over a finite time horizon. The first command in the sequence is then used as the control action for the plant and the optimization process repeats to determine a new control. Feedback and disturbance rejection are incorporated into the technique by updating state estimates at a discrete time interval and executing the optimization procedure over the newly shifted time horizon. Figure 3, presents a visualization of the general RHC process, where an optimal input control sequence produces a minimal cost state trajectory in the state-space, when predicted though the system dynamics function of equation (3.1).

A key concept of RHC is that the state trajectory being tuned is generated by predicting the system's future states through a dynamical model. The procedure determines a set of open-loop

commands which, when extrapolated through this model, yield a minimum cost path over the finite time horizon.

Remark 1: The use of RHC for an AGV inherently unifies the planning and control tasks, because the vehicle's future motion is continually being optimized as a sub-problem of the overall control task. This continuous online optimization is equivalent to a separate planning process.

The optimal control problem in receding horizon control is usually posed as the minimization of a quadratic cost function over the time interval $[t, t+T]$. For a time-invariant nonlinear system, the optimization is subject to the dynamics, which are given is their discrete form as,

$$x(t+1) = f\left(x(t), u(t)\right), \tag{3.1}$$

where $x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m$ denote the unconstrained state and control input vectors respectively. The dynamics function $f(\cdot): \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, is assumed to be continuous at the origin with $f(0,0) = 0$. The cost / value function for the optimization problem is typically given as

$$V(x(t), u(t)) = \sum_{\tau=t}^{t+T} x^T(\tau) Q x(\tau) + u^T(\tau) R u(\tau), \tag{3.2}$$

here both Q and R are positive-definite symmetric constant weighting / gain matrices. The optimal control task is to choose a sequence of future controls, which minimize the cost function over the projected time interval. The optimal input sequence is given by:

$$\pi_t = \left\{\hat{u}_t, \hat{u}_{t+1}, \hat{u}_{t+2}, \cdots, \hat{u}_{t+T-1}\right\}. \tag{3.3}$$

This optimal control sequence thus has a corresponding sequence of future states, which is governed by (3.1). The instantaneous control input to the system in RHC is then selected as the first control vector in the sequence π_t:

$$u(t) = \hat{u}_t . \tag{3.4}$$

Repeating this process thus yields a closed-loop control system, because upon every time step the current state information is updated and a new optimal control is computed. Thus by repeatedly updating the state information, a feedback mechanism is inherently introduced into the control scheme.

By substitution of the optimal control and predicted state sequence into equation (3.2), the minimum cost for the given state x is established. Typically RHC is used to control constrained nonlinear systems where: $x(t) \in \mathbb{X} \subset \mathbb{R}^n, u(t) \in \mathbb{U} \subset \mathbb{R}^m$ are the constrained input and state-space which are convex subsets of the unconstrained spaces, which include the origin at their interior. As a required "ingredient" for stability, a terminal state constraint $x(t+T) \in \Omega$ is also included in the problem, where $\Omega \subset \mathbb{X} \subset \mathbb{R}^n$ is a convex subset of the state-space which also includes the origin at its interior. (The exact conditions which must be met for RHC stability are differed to later in the chapter.) With these constraints, the optimal cost function for any given state can be expressed as the optimization problem solution:

$$V^*(x) = \min_{u[t,t+T-1]} \sum_{\tau=t}^{t+T} x^T(\tau) Q x(\tau) + u^T(\tau) R u(\tau)$$
$$\text{subject to } \begin{cases} x(t+1) = f\left(x(t), u(t)\right) \\ x(t) \in \mathbb{X} \\ u(t) \in \mathbb{U} \\ x(t+T) \in \Omega \end{cases} . \tag{3.5}$$

Notice that equation (3.5) is explicitly independent of time. This is because the underlying optimization of the system is time-invariant. In other words the optimization problem will always yield the same result for any specific state regardless of time.

Generally the minimal cost and its associated optimal control sequence are not known ahead of time and therefore a mechanism to determine these values is required to implement a receding horizon controller. Most commonly, the optimization problem in receding horizon control is solved with Quadratic Programming techniques (due to the classically quadratic nature of the cost function) or Mixed Integer Quadratic Programming (MIQP) [BEM00] (for finite and discrete systems). These are computationally expensive, complicated, and time consuming processes, which limit the use of receding horizon control for electro-mechanical systems. This limitation is primarily due to the time critical nature of the required control, i.e., most electro-mechanical system have relatively short time constants and time-to-double instability criteria. (Time-to-double is a widely used metric used to describe the amount of time it takes for an unstable system to double in amplitude. The shorter the time-to-double metric is, the more unstable the system is, and therefore it is more difficult to control.) However, RHC has seen the most of its success in the chemical process industry where these types of system parameters can be several orders of magnitude larger than their electro-mechanical counterparts, and therefore a slower computation of the optimal control is acceptable.

Motivated by the need to solve for these complex optimal trajectories online for electro-mechanical systems, the proposed heuristic receding horizon control (HRHC) uses a finite graph search known as A* (A-Star) to find the optimal control sequence, rather than a dynamic programming approach. A* search is a technique originally developed for Artificial Intelligence

applications; however the method lends itself elegantly for efficiently solving complex searches in state-space defined systems.

Furthermore, since the optimization problem can become much more difficult for a non-quadratic cost function, in which systems are more generally defined with the value function:

$$V(x) = \min_{u[t,t+T-1]} \sum_{\tau=t}^{t+T} L\left(x(\tau), u(\tau)\right), \tag{3.6}$$

where $L\left(x(t), u(t)\right)$ is the general nonnegative intermediate cost function over the optimization interval. A* search can be applied to such a system with little impact attributable to the complexity of $L(\cdot)$. Other optimization methods require the function $L(\cdot)$ to be in a specific form (primarily quadratic).

One essential requirement of A* however, is that the state-space and input space be discrete and quantized. (Generally it is more important to have a quantized input space. The state-space may remain continuous, so long as a few simple modifications are made to the search. This will be discussed in more detail in the third section.) Classically, receding horizon control requires that the input and state-spaces are continuous in order to guarantee stability and optimality [MAY90] (Solutions that achieve these properties exist for both discrete and continuous systems. See [NIC98]). To apply A* to the RHC problem the notion of both stability and optimality must therefore be modified to include results that are sufficient but nevertheless sub-optimal. Researchers have shown that sub-optimal solutions to the classic RHC problem can maintain overall stability [SCO99], but the idea of stability changes slightly when considering a system controlled by a quantized input set [BRO00].

It should be noted that the consideration of quantized input control is inherently more robust for implementation on real-world systems for a very practical reason. The modern

approach for control systems incorporates the use of discrete computers and their respective electronic interfaces to the continuous system. The underlying control mechanisms are thus discrete and quantized, i.e. program variables in memory, analog to digital converters, etc. Therefore the inputs to the continuous systems are also discrete and quantized. Although these facts are more often now neglected due to the increasing resolution and speed of computing and interface hardware, by considering their effects in the control solution, a better defined and more predictable behavior can be achieved.

The remaining portions of this chapter are broken down into sections as follows: in the second section, basic assumptions, notation and preliminary results are discussed. The third section defines the A* algorithm used by HRHC. In the fourth section, the required quantization and invariant set properties needed for stability criteria are given, and the formulation of the novel HRHC is presented in the fifth section. Dual Frequency Receding Horizon Control (DFRHC) is shown in 6, and finally conclusions are presented.

Notation, Assumptions, and Preliminary Theorems

An essential theorem for RHC is that of the suboptimal stability conditions. When met, these criteria provide proof that the system will maintain stability, and converge to a desired set point. Classically, the stability of nonlinear systems has been identified with the standard Lyapunov stability theorem. One fundamental requirement of this theorem is that the input control u of the system is a continuous function of the state x, and hence this also implies that the control for any particular state be unique.

The suboptimal stability results are formulated in a manner similar to that of the standard Lyapunov stability theorem; however they allow for a non-unique and discontinuous control law. As will be shown later in the chapter this is the case that must be considered for HRHC, because of the finite and quantized nature of the A* search.

Before the suboptimal stability theorem is presented, some basic notation must first introduce. First, the Euclidian norm of vector *x* is denoted as $\|x\| \in \mathbb{R}$ where the dimensionality of the vector x, is identified through the context. Any function $\alpha(\cdot)$ defined on the range $[0,\infty)$ is considered a class *K-function* if it is continuous and strictly increasing, with $\alpha(0) = 0$. Lastly, let B_r^n denote a closed *ball* set of radius *r* in the space $\mathbb{R}^n$, or in another form, let the set $B_r^n := \{x \in \mathbb{R}^n \mid \|x\| \le r\}$.

With these concepts defined, the suboptimal RHC stability theorem is referenced from [SCO99], and provides the basis of stability for the newly proposed HRHC scheme of this dissertation.

Feasibility Theorem for Suboptimal RHC from [SCO99], let there exist:

1) a value function $V(\cdot): \mathbb{R}$ which is continuous at the origin with $V(0,0) = 0$ and a *K-function* $\alpha(\cdot)$, such that

$$V(x(t)) \ge \alpha(\|x(t)\|) \; \forall x \in \mathbb{R}^n \tag{3.7}$$

2) a set $X_0 \subseteq \mathbb{R}^n$ that contains an open neighborhood of the origin and a *K-function* $\gamma(\cdot)$, such that every realization of the controlled system with $x(0) \in X_0$, satisfies $x(t) \in X_0$ for all $t \ge 0$ and

$$V(x(t+1)) - V(x(t)) \le -\gamma(\|x(t)\|) \tag{3.8}$$

3) a constant $r > 0$ and a *K-function* $\sigma(\cdot)$, such that every realization of the controlled system with $x(t) \in B_r^n$ satisfies

$$\|\pi_t\| \leq \sigma\left(\|x(t)\|\right). \tag{3.9}$$

Then the controlled system is asymptotically stable in $X_0 \subseteq \mathbb{R}^n$.

This theorem simply identifies that: if the value function can be lower bounded by a class *K-function*, and if the change in the value function can be upper bounded by another *K-function*, and the norm of the suboptimal control sequence $\|\pi_t\|$ can be upper bound by a *K-function*, then the controlled system will be asymptotically stable in a the local region X_0. The reader is referred to text [SCO99] for the complete detailed proof of the theorem.

A* Algorithm and Admissibility

As stated in section I, the process of determining the open loop input sequence (3.3) requires an optimization method in the receding horizon control scheme. One way to accomplish this step is to conduct a search for the optimal sequence over a finite input and state-space graph. One of the most readily used techniques to do this is known as the A* (A-Star) search. The bulk of information provided in this section is given in [NIL71] and [NIL98]. This gives a synopsis of the provided formulations and is used only as an introduction of the required knowledge needed to detail HRHC. For a more complete discussion refer to [HAR68].

A* is an admissible graph search algorithm. The admissibility of the algorithm guarantees that it will always terminate with a minimum cost optimal solution sequence if one exists. This is a required property for implementation of receding horizon control since the optimality of a solution is closely related to the stability of the control. The A* algorithm is a heuristically guided search. The term heuristic means serving to aid to discover, and is derived from the Greek verb *heuriskein*, meaning “to find”. The heuristic information, as will be seen, depends on some educated or special knowledge about the type of problem being represented by the graph. This

information is used to guide the search in order to increase its efficiency and timeliness. The heuristic information is represented by a heuristic function in the search algorithm. It will be shown that if this function adheres to some simple constraints the search can be made to be both efficient and admissible.

Before the A* algorithm is discussed, it is first necessary define the general graph-searching process. A graph consists of a set of *nodes*, where each node represents a particular configuration in a corresponding state-space. A node in the graph can be connected to another node in the graph by an *arc*. The arc defines a way to get from one node to the other and it can be correlated to a unique transition in the represented state-space.

A graph search is a process that determines a sequence of state transitions, represented by arcs in the graph, that allow for the transversal from one specific node to another specific node or group of nodes. Every search therefore has a *start node*, which is associated with the initial state configuration, and a set of *goal nodes* or single goal node, which represent a final desired configuration.

The successors for any node in the search, i.e. the set adjacent nodes connected to the original node by a single arc, are calculated via operations applicable to the corresponding state configuration describe by that node. For example, for a nonlinear system defined by (3.1), any state configuration x_t, represented by node n_{x_t}, can by succeeded by a set of states X_{t+1}, which are the result of applying the set of possible input commands U_{x_t} over a finite period of time. A way to show this process is as a mapping from the given state to the set of future possible states, as seen here:

$$x_t \xrightarrow[u \in U_x]{f(x_t,u)} X_{t+1}. \tag{3.10}$$

This operation is called the *expansion* of a node and is denoted by the operator $\Gamma(n)$. The expansion of a node therefore produces a set of successor nodes, which can be shown with:

$$N_{i+1} = \Gamma(n_i). \tag{3.11}$$

As a direct requirement of the expansion process, when applying any graph search to a linear or non-linear state-space control system, the input space (u) must be finite and discrete (thereby implying a quantize-input system). Without a discrete input-space, the expansion of a single node would generate an infinite number of successor nodes. Clearly this would result in an undefined search process. By using a naturally or artificially quantized input space, the graph search remains finite and computable. This quantization has a profound impact on the overall control stability, the complete discussion of which is deferred to later in the chapter.

During the expansion process, *pointers* are set up from each successor node to its parent. These pointers are used to trace a route back to the start node once a goal node has been found. The functions which define the relationship of a node with its represented state and the pointer to its parent are given here:

$$n_i = node\left(x(t), u(t), n_{i-1}\right) \tag{3.12}$$

$$\begin{aligned} xnode(n_i) &= x(t) \\ unode(n_i) &= u(t) \\ pnode(n_i) &= n_{i-1} \end{aligned} \tag{3.13}$$

where equation (3.12) is the node construction function, which requires a state, input and predecessor node, and the equations in (3.13), provide a means to access a given node's state, control input, and parent node.

The graph structure automatically created as a result of the expansion process is known as a *tree*. A tree is a graph in which each node has a unique parent, except for an origin node which has no parent, and is called the *root node*. In this case the root node is clearly the start node. Trees also have the unique property in that every path to a node in the graph is unique. Therefore, it can be guaranteed that each node created in the tree has never been generated before nor will it be generated again.

As the expansion process continues, each of the newly generated successors are checked to see if any of their represented state configurations meet the required goal state criteria. If such a goal node is found the arcs connecting the solution nodes are traced back to the start using the pointers, and the corresponding solution sequence of state operators, which produce the path to the goal, is generated and returned as the result.

This description is of a general graph search process. However, for the description to be complete, another process which clearly defines the order in which nodes are expanded must be established. One way to define this process is simply to expand nodes in the order in which they were generated; this is known as a *breadth-first* search. Another way is to expand the nodes which were most recently generated, a process called *depth-first* search. Both breadth-first and depth-first are known as *blind-search* procedures, because they do not use any information which is relevant to the problem being represented in the search graph.

A* search uses heuristic information to provide a more informed way to search through the graph and its represented state-space. The heuristic information is given in the form of a mathematical cost estimation function, which is part of an *evaluation function* that is used to order the node expansion process. This evaluation function then serves the purpose of ranking candidate nodes by which one is most likely to be on the best path to the goal.

The A* algorithm defines the evaluation function $\hat{f}(n)$ as an estimate of the cost to reach the goal along a path which is constrained to pass through node *n*. This evaluation function therefore estimates the cost of a minimal cost path, as the sum of the estimated cost from the start node *s* to *n*, and the estimated minimal cost from the node *n* to any goal node. The candidate node which has the minimum value of $\hat{f}(\cdot)$ is thus the node with the best chance of being on the minimal cost path to the goal, and it should be expanded next in the search.

Assuming the function $k(n_i, n_j)$ provides the true minimum cost from node n_i to node n_j, the following cost function is defined:

$$h(n) = \min_{g \in G} k(n, g). \tag{3.14}$$

Here *G* is a set of goal nodes. Note that this function is undefined for any node from which the set of goal nodes is unreachable. The function

$$g(n) = k(s, n), \tag{3.15}$$

provides the minimum cost from the unique start node to the node *n*, and is only defined for any node in which a path from the start node exists. The sum of equations (3.14) and (3.15) is defined as

$$f(n) = g(n) + h(n), \tag{3.16}$$

and is the true minimum cost from the start node to the goal, on a path constrained to pass through the node *n*.

Since A* requires the evaluation function to be an estimate of (3.16), the estimate is defined as

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n), \tag{3.17}$$

where $\hat{g}(\cdot)$ is an estimate of $g(\cdot)$ and $\hat{h}(\cdot)$ is an estimate of $h(\cdot)$. Clearly $\hat{g}(n)$ can be calculated by simply summing the accumulated costs of single arc transitions from the start node through any successor nodes and ultimately to the node *n*. The estimate $\hat{h}(\cdot)$ however is much more difficult to calculate since no future knowledge of the minimum cost path to the goal exists, because such a path is constructed only when the search is finished. At a midpoint in the search, heuristic information or knowledge about the problem being represented must be used. Hence $\hat{h}(\cdot)$ is called the *heuristic function*. This heuristic function must meet certain criteria in order to maintain admissibility of the A* algorithm.

The A* algorithm is defined as an extension to the general graph search process, which uses the evaluation function given in (3.17) to select the order in which nodes are expanded. The algorithm is broken down into a sequence of steps shown in Table. Essentially, the search is conducted via the management of two sets, one set *O*, is called the open set, and the other set *C*, is known as the closed set. The open set maintains a list of candidate nodes which have not been checked to see if they exist in the goal set. The closed set contains all of the previously expanded nodes, which are not in the goal set, however must still be managed in order to allow a solution sequence to be traced back to the start node from the goal node.

Initially the algorithm is given the start node *s*, and the goal set *G*. The search continues the process of finding a minimum cost estimate node on the open set, and expanding the nodes until either a solution node is found or the open set runs out of candidates. The expansion operator $\Gamma(\cdot)$ is responsible for generating successor nodes (if they exist) and setting up pointers to the

parent node. If a goal node is found, it is returned as the solution and it can be traced back to the start node through the established pointers.

Up until this point in the chapter, it has only been stated that the evaluation/cost function, $\hat{f}(n)$, must meet certain conditions in order for the A* search to find an admissible solution sequence to the goal set. This fact is based off of two keystone theorems proved by the creators of the A* algorithm [HAR68]. They are cited here only for reference. The reader is referred to the previous source for detailed proofs.

A* Lemma: If $\hat{h}(n) \le h(n)$ for all *n*, then at any time before A* terminates and for any optimal path *P* from node *s* to a goal, there exists an open node n' on *P* with $\hat{f}(n') \le f(s)$

This lemma establishes that if the heuristic function cost estimate is less than the true cost to the goal for all nodes, then some node n', created during the A* search process, exists on the set of open nodes, and its corresponding path cost estimate $\hat{f}(n')$ is less than the true minimum cost path to the goal from the start node *s*.

A* Theorem: If $\hat{h}(n) \le h(n)$ for all nodes *n*, and if all arc costs are greater than some small positive number δ, then algorithm A* is admissible.

The initial A* lemma is used to prove the primary A* theorem which states that if a heuristic function is chosen that always underestimates the true cost to the goal, and if state transitions along arcs between nodes have some non-zero cost, then the A* algorithm will always terminate in an admissible solution if one exists.

For the theoretical approach described here, A* is used as a replacement for a classical optimization routine, and therefore an algorithm is desired that can obtain an admissible, and thus near optimal, solution sequence from a start state to a goal state. In order to adhere to the A*

theorems, a heuristic function is chosen that always underestimates the state transition cost to the goal region, in order for this implementation of RHC to be correct. The meaning of this heuristic function and how it is selected is very important and is the basis of the Heuristic RHC implementation discussion in the next chapter.

Quantization and Invariant Sets

As aforementioned in the previous section, in order to use the A* algorithm in place of an input sequence optimizer, a quantized (artificially if need be) input space must be used for the system that is being controlled. This is because the expansion of nodes during the search must yield a finite set of successor nodes rather than an infinite continuum of possible future states. Since the expansion of a node is calculated by stimulating all possible input commands over an interval of time, then the number of possible commands must also be finite and hence quantized. This quantization will guarantee a finite set of successor states for any given configuration represented by a node. The quantized input set $\mathcal{U}$ is a subset of the continuous real input space and is defined as:

$$\mathcal{U} := \{u_1, u_2, \ldots u_N\} \subset \mathbb{R}^m, \tag{3.18}$$

where subscripts 1 through *N* denote the different levels of quantization.

The discussion of the effects of input quantization first requires some critical definitions which are now introduced before the following formulations of the chapter:

Definition 1: A set of state-space configurations is considered to be *zero-measure* set if it is made up of only singular discontinuous configurations, such that the measurable probability of finding the system in that state would be equal to zero. Example: in a two-dimensional state-space, a non-null zero-measure set would consist only of points or curves in the configuration

space, whereas a measurable set would consist of at least one continuous area or region. Likewise, the three-dimensional equivalent would require some measurable volume.

Definition 2: The terminal constraint region $\Omega \subset \mathbb{R}^n$ is a *controlled-invariant* set for system (3.1) iff $\forall x \in \Omega \; \exists \, u \in \mathcal{U}$ such that $x(t+1) = f(x,u) \in \Omega$.

Definition 3: If $\Omega \subset X_0 \subseteq \mathbb{R}^n$, then the set Ω is said to be $X_0 - attractive$ iff $\forall x \in X_0$ there exists some trajectory contained in X_0 that enters Ω in a finite number of steps. In addition, if $\forall x \in X_0$ the trajectory can be chosen to be of length T (where T is the number of steps in the time horizon), then Ω is said to be $X_0 - attractive\ in\ T\ steps$.

By Definition 1, it can clearly be seen that the quantized input set is indeed a zero-measure set in the constrained input space. The consequences of a quantized input space belonging to a continuous nonlinear system, such as an Autonomous Ground Vehicle, are far reaching in their impact on its stability and its reachable state configurations. The classic concept of stability, i.e. asymptotic or exponential convergence to some set-point or tracking point, is lost because the zero-measure nature of the control prevents the system from performing arbitrary state transitions. Instead, the system is restricted to a finite set of future states, which depend on the levels of quantization and the number of decision steps or time horizon. This restricted performance requires a more "practical" sense of stability, in which the system is allowed to converge to a broad set-region in the state-space. This notion was made clear in [DEL90] for an attempt to define more realistic stability properties for quantized systems.

Also, since the input space is quantized, the determined solution will always be sub-optimal. The reason for this is that the true optimal solution is assumed to be a sequence, which exists somewhere in the continuous input space. Since the input set is zero-measure, the probability of finding the true optimal input sequence within the quantized set is zero. In other

words, a quantized solution cannot include every possible input and therefore one will never practically be able to find the true minimum cost trajectory that exists somewhere in the continuous state-space. This is reflected in a comparison of the optimal state transition cost between the quantized input system and the continuous input system:

$$\min_{u \in \mathcal{U}} V(x,u) \geq \min_{u \in \mathbb{R}^m} V(x,u). \tag{3.19}$$

Figure shows the possible input space sequence over the time horizon, and that the optimal solution values will never exactly touch the possible quantized input values. As a result, the optimal quantized input state-space trajectory will differ from the continuous input optimal state trajectory. This trajectory difference will result in a larger value function cost, and thus is considered to be suboptimal.

In order to insure stability of a RHC, one solution [MAY90] is to impose a terminal state constraint, $x(t+T)=0$ on the optimization problem. For a quantized input system, this would result in a zero-measure set of possible initial states, thus rendering this stability constraint unrealistic. Rather a quantized input system must incorporate a relaxed terminal state constraint to include an entire controlled-invariant region Ω. This allows the optimization problem to find a viable solution for a feasible and significant set of initial states.

The significance of Definition 3 is that the region of attraction for the terminal state region Ω is only guaranteed to be local within the region X_0. That is to say the system will only stabilize when initialized within the region X_0. This concept is identical to that described in the feasibility theorem presented in the second section.

Heuristic Receding Horizon Control

Motivated by the desire to unify the planning and control tasks for an AGV, Remark 1 proposes RHC as an elegant solution to combine the two problems into one. Unfortunately, traditional methods of optimization required for RHC hamper the ability to use the technique for electro-mechanical systems. This is due to the time consuming nature of the online optimization, which is usually much longer than the system response time. As a means to greatly reduce online computation, this dissertation proposes the idea of Heuristic Receding Horizon Control (HRHC). In HRHC, heuristic information about the problem is used to maximize the performance of an online search of the system's input and state spaces. This search is then substituted in place of the usual optimization routines.

A proven algorithm for heuristic searching in a finite decision space is the A* search. Section II of this chapter has presented the formal basis for the algorithm's admissibility and its general computation. The direct consequence of using the A* search for RHC is that a quantized input space must be used for this system. This quantization in turn has further consequences on the performance and stability of the system, which must now be characterized in a different form when compared to classic stability metrics. These effects have been detailed in Section III, and the critical definitions and concepts presented in that section are now required for the complete stability analysis of HRHC.

To begin, the general cost optimization problem, subject to the terminal state constraint is revisited, with the added constraint that the input set be that of the quantized input space $\mathcal{U}$. Thus the RHC optimization problem now takes on the form:

$$V^*(x) = \min_{u[t,t+T-1]} \sum_{\tau=t}^{t+T} L\left(x(\tau), u(\tau)\right)$$
$$\text{subject to } \begin{cases} x(t+1) = f\left(x(t), u(t)\right) \\ x(t) \in \mathbb{X} \\ u(t) \in \mathcal{U} \\ x(t+T) \in \Omega \end{cases}. \tag{3.20}$$

Furthermore, the A* minimum node-to-node transition cost function $k(\cdot)$ is also defined such that it is equivalent to that of the RHC general cost function given in (3.6), by taking advantage of the node to state relationship functions provided in (3.13):

$$k\left(n_i, n_j\right) = \min_{u\left[n_i, n_j\right]} \sum_{k=i}^{j} L\left(xnode\left(n_k\right), unode\left(n_k\right)\right)$$
$$subject\ to \begin{cases} n_{k+1} \in \Gamma\left(n_k\right) \end{cases}. \tag{3.21}$$

Also the expansion function Gamma is more completely defined as the union of all generated nodes over the quantized input set U and extrapolated though the dynamics function $f(\bullet)$, thus yielding:

$$\Gamma\left(n_i\right) = \bigcup_{u \in \mathcal{U}} node\left(f\left(xnode\left(n_i\right), u\right), u, n_i\right)$$
$$subject\ to \begin{cases} f\left(xnode\left(n_i\right), u\right) \in \mathbb{X} \end{cases}. \tag{3.22}$$

The combination of the two techniques used in HRHC, namely the A* search algorithm and Receding Horizon Control, is based off of the key concept that the general value function presented in (3.6), is obviously related to the A* cost-to-go function, which is given in (3.14).

Lemma 1: The RHC optimal value function $V^*(x(t))$ subject to the quantization constraints $u \in \mathcal{U}; x(t+T) \in \Omega$ is equivalent to the A* cost-to-go function $h\left(n_i\right)$ when the

function $k\left(n_i, n_j\right)$ is defined as in (3.21) with the additional condition that the terminal state $x\left(t+T\right)$ is represented by node n_j and $\forall g \in G \mid xnode(g) \in \Omega$.

Proof: Given that $xnode\left(n_k\right) = x\left(\tau\right)$, implies node n_k represents the system state at arbitrary time τ. Since $h\left(n_k\right)$ is the minimum cost from node n_k, to the goal region *G*, via the expansion operator $\Gamma\left(\cdot\right)$, then the cost accumulated over each state transition arc to the goal will equal that of the identical state transition sequence calculated via (3.6), and since $\Gamma\left(\cdot\right)$ is constrained to expand only quantized inputs through the mapping (3.10), then the state transitions will be restricted to the quantized permitted inputs used in the optimization problem(3.20). Therefore, the A* cost-to-go function is written more completely by the equations:

$$\begin{aligned} h\left(n_i\right) &= \min_{n_j \in G} \min_{u\left[n_i, n_j\right]} \sum_{k=i}^{j} L\left(xnode\left(n_k\right), unode\left(n_k\right)\right) \\ subject\ to &\left\{ n_{k+1} = \Gamma\left(n_k\right) \right. \end{aligned} . \tag{3.23}$$

It can be deduced from this nested minimization of the cost function that the overall minimum cost via a set of node input commands $u\left[n_i, n_j\right]$, will be the same for a single minimization subject to the additional constraint $n_j \in G$. Thus the nested minimization can be written as a single minimization in the form:

$$\begin{aligned} h\left(n_i\right) &= \min_{u\left[n_i, n_j\right]} \sum_{k=i}^{j} L\left(xnode\left(n_k\right), unode\left(n_k\right)\right) \\ subject\ to &\begin{cases} n_{k+1} = \Gamma\left(n_k\right) \\ n_j \in G \end{cases} \end{aligned} \tag{3.24}$$

In addition, if j is chosen such that $xnode\left(n_j\right) \equiv x\left(t+T\right)$, or in other words all terminal nodes at generation j represent the system at time $t+T$, and since the expansion operator is constrained to quantized input extrapolations through the dynamics function (3.1), and limited to the constrained state-space $\mathbb{X}$, then the constraints of (3.24) are identical to (3.20), and thus

$$h\left(n_i\right) = V^*\left(x\left(t\right)\right). \tag{3.25}$$

Theorem 1: If the A* heuristic function $\hat{h}\left(n_i\right)$ is selected such that $\hat{h}\left(n_i\right) \le V^*\left(x(t)\right)$ then the algorithm will terminate with a solution node, which when traced back to the start node represents an input decision sequence with the equivalent optimal value function V*(x) subject to the constraints of equation (3.20).

Proof: From lemma 1, it is clear that the optimal value function and the A* cost-to-go functions are equivalent. The proof of Theorem 1 then follows from the A* admissibility Theorem referenced in the third section. If the heuristic function is selected such that $\hat{h}\left(n_i\right) \le V^*\left(x(t)\right)$, then obviously $\hat{h}\left(n_i\right) \le h\left(n_i\right)$, and therefore by the A* Theorem it is known that the algorithm will terminate with an admissible solution, and so the corresponding state transition sequence will be the equivalent of one determined with the quantized and constrained optimization problem of (3.20).

The stability criteria of HRHC follow that of the suboptimal RHC (Feasibility Implies Stability) theorem referenced in Section II. Since the A* quantized optimal solution sequence has to be suboptimal compared to the continuous input optimal solution, then the conditions that HRHC must meet in order to maintain system stability must be relaxed to those of the suboptimal requirements. The first condition of the suboptimal RHC theorem is satisfied by the design of the cost function, which can be chosen such that it is lower bounded by a class K-

function. The second condition is also easy to satisfy by imposing an additional constraint on the search whereby the cost values along the solved trajectories must decrease by at least $\mu L\left(x\left(t\right),u\left(t\right)\right)$ where $\mu \in \left(0,1\right]$ is just a constant. Lastly the third criteria is usually satisfied by assumption, however as will be discussed in the next chapter this constraint is less important for quantized systems where one can find some controlled-invariant set Ω, or even less important, when the system is simply turned off in the terminal state region. The main purpose of HRHC is to find a fast solution that will drive the system towards the goal region. With parts one and two of the feasibility theorem met, that is all one needs to do the job.

An outline of the HRHC algorithm is given in Table 3-2. The basic process is to first generate the A* start node based off of the system's current state, input and a null parent node pointer. Then an A* search is conducted, in order to find the least cost path from the start node to the goal region. The search differs slightly from the one outlined in Table, in that the goal test checks the state represented by the node for membership in the goal region set Ω. Also, the solution node is not returned rather it is traced back to find the initial optimal input command in the sequence. If a solution doesn't exist then the controller faults and external techniques could take corrective action.

The outlined algorithm only represents a single iteration of the HRHC control loop, and therefore the process is repeated online at a fixed interval, with state feedback information and any changes to the goal set if they exist.

Dual-Frequency Receding Horizon Control

For systems capable of providing feedback information at a rate higher than that required for predictive planning and control, it may be desirable to take advantage of the feedback in real-time, rather than postponing any state updates until the next RHC optimization iteration.

Furthermore, since RHC requires the prediction and optimization time steps to equal that of the state update period, if the update frequency is very high, then too many intermediate steps to the desired horizon time may be required.

The purpose of Dual-Frequency Receding Horizon Control (DFRHC) is to allow predictive optimization out to a desired time horizon, while simultaneously reducing the total number of planning steps and integrating feedback information at its real-time rate. The method works by predicting the system state response through a series of constant input commands, much in the same way classic RHC works. The difference is that the constant input commands are held over a time step period which is a multiple of the shorter feedback period. For example, if the feedback period is one millisecond, then the optimization prediction period may be five milliseconds. This lower frequency prediction sequence is then optimized and the first command in the sequence is executed for one millisecond. Then the process repeats, by again predicting the system's response through a chain of five millisecond input commands out to the time horizon. Figure diagrams the general DFRHC process. Shown there is a three step planning sequence out to the time horizon T, where the prediction period is denoted as p. This planning period p is just some constant integer greater than the feedback period, which is just one for the example. The first input in the control sequence is then applied to the system for one feedback cycle, and the lower frequency optimization process is repeated to determine a new command.

The name Dual-Frequency, comes from the fact that the optimization prediction steps are planned as if they are to be executed at one frequency, when actually only the first command in the sequence is used as the system input for the feedback cycle, which is operating at a second fixed frequency.

This method has several advantages, when compared to classic RHC. One of the benefits is the ability to maintain optimization computability in real-time, over an extended time horizon. The reason for this is that since the time of computation increases exponentially with the number of planning steps, by reducing the number of steps out to the horizon one can decrease computation. Because of this, the method also allows for a longer time horizon, which increases the region of attraction for the controller, and allows for a larger envelope of system operation. Another important lead is that maintaining feedback at a high rate, allows for faster disturbance rejection, and increases robustness with respect to model inaccuracies, because of the system's ability to react to unpredicted changes faster.

A disadvantage however is that because there are fewer planning steps, any obtainable state trajectories are more constrained. Therefore, this will result in a higher cost value function, and will produce results less optimal than a high frequency input sequence.

An illustrative example that explains the strategy behind DFRHC can be presented in the form of an investment planning problem. Imagine for example a scenario where some investor is planning a five year investment policy. The classic RHC method would perhaps have the plan call for purchases and sales once a year and these transactions would be planned and predicted over what the investor thought the market was going to do for the next five years. However, if there were a market crash at the midpoint of the year, the initial strategy would not allow for rapid transactions to correct for the unforeseen crash. The investor would have to wait for the next year in order to plan and execute new transactions.

A better strategy would be to plan transactions over the long term, and reevaluate those transactions on a frequent basis. To reduce complexity, the investments would not be planned at the same frequency as the reevaluations, but would rather represent only a broad strategy over

the five year period. This way, if a market crash did happen, the frequent reevaluations would allow for a rapid correction in the long term plan. This is the essence of DFRHC, changes to the plan can happen fast as new information becomes available, and this way if a large unpredicted event happens it can be accounted for right away.

The computation process of DFRHC is nearly identical to the classic RHC with the exception that the intermediate predicted states are calculated recursively through the dynamics function (3.1), over each individual planning step. If the planning period is specified as *p* state update periods the calculation of the a future state $x(t+p)$ given a constant input u is

$$x(t+p) = f_1\left(f_2\left(\ldots f_p\left(x(t),u\right),u\right),u\right), \tag{3.26}$$

where subscripts 1 through *p* simply represent the iteration of the recursively calculated state update function.

It should also be noted that DFRHC is very easily incorporated into HRHC since the only change to the calculation is that the predicted states must be calculated recursively as in function (3.26). This requires only a trivial change to the node expansion function.

Conclusions

This chapter has introduced two new and novel RHC advancements. The first is HRHC, in which the general strategy is to employ heuristic information about the control problem being represented in order to maximize the performance of the required online optimization problem. This is done by replacing any usual optimization methods, which are often too slow for electro-mechanical systems, with an A* search algorithm. This algorithm is designed to take advantage of heuristic information during the search process for any system defined in a state-space.

The second technique is DFRHC, where the general concept is to optimize a state trajectory over a set of extended period planning steps. The steps are planned at a frequency

lower than that of the feedback update frequency, and then the process repeats after a single feedback iteration.

Both methods are easily combined together, which allows for a combined effect that further increases optimization performance to an extended time horizon. This combination is simple to do because the expansion operation in the A* search is simply changed to extrapolate state trajectories over a repeated single input command.

One very important item not discussed in this chapter is the ability to use RHC methods to plan AGV motion through an obstacle riddled environment. As per Remark 1 it should be possible to use RHC to execute simultaneously planning and control. However, in order to incorporate motion optimization around obstacles or though some other complex structured environment, then some non-trivial changes must be made to the setup of the RHC problem. These changes and their effects on system stability and performance are the focus of the following chapter.

Table 3-1: The algorithm AStarSearch is used to find an admissible solution sequence to a goal region from a given start node. It requires the definition of an admissible cost function, and expansion process, in order to successfully execute.

Line	Action	Comment
1:	**Algorithm AStarSearch(s, G):**	
2:	$O = s$	// Let the open set equal the start node
3:	$C = \varnothing$	// Let the closed set equal null
4:	*while* $O \neq \varnothing$ *do*	// While the open set is not empty
5:	$n = \underset{i}{\operatorname{argmin}}\ \hat{f}\left(O_i\right)$	// Find the minimum cost estimate node on open
6:	$O = O - n$	// Remove the node from open
7:	$C = C \cup n$	// Put the node on closed
8:	*if* $n \in G$ *then*	// If the node is a member of the goal set
9:	*return n*	// Return the solution node
10:	*endif*	// End of if statement
11:	$O = O \cup \Gamma\left(n\right)$	// Expand the node and put successors on open
12:	*endwhile*	// End of while loop
13:	*return* $\varnothing$	// Return no solution found

Table 3-2: This table represents the algorithm for a single HRHC iteration. This process is repeated online at a fixed rate, with feedback state, input and goal information.

Line	Action	Comment
1:	**Algorithm HeuristicRHC(x, u, Ω):**	
2:	$s = node(x, u, \varnothing)$	// Initialize start node to current state and control
		// Begin to conduct an A* search
3:	$O = s$	// Let the open set equal the start node
4:	$C = \varnothing$	// Let the closed set equal null
5:	*while* $O \neq \varnothing$ *do*	// While the open set is not empty
6:	$n = \underset{i}{\operatorname{argmin}}\ \hat{f}(O_i)$	// Find the minimum cost estimate node on open
7:	$O = O - n$	// Remove the node from open
8:	$C = C \cup n$	// Put the node on closed
9:	*if* $xnode(n) \in \Omega$ *then*	// If the node state is a member of the goal set
10:	*endwhile*	// A solution has been found, so end the search
11:	*endif*	// End of if statement
12:	$O = O \cup \Gamma(n)$	// Expand the node and put successors on open
13:	$n = \varnothing$	// Reset the node to null
14:	*endwhile*	// End of while loop
		// Now trace back the solution
15:	*if* $n \neq \varnothing$ *then*	// If a goal node has been found
16:	*while* $p(n) \neq \varnothing$ *do*	// Loop back until the start node is reached
17:	$n = p(n)$	// Set the current node equal to its parent
18:	*endwhile*	// End of while loop
19:	*return unode(n)*	// Return the initial optimal control input
20:	*else*	// Else no solution has been node found
21:	*return* $\varnothing$	// Return null to indicate a fault
22:	*endif*	// End of if statement

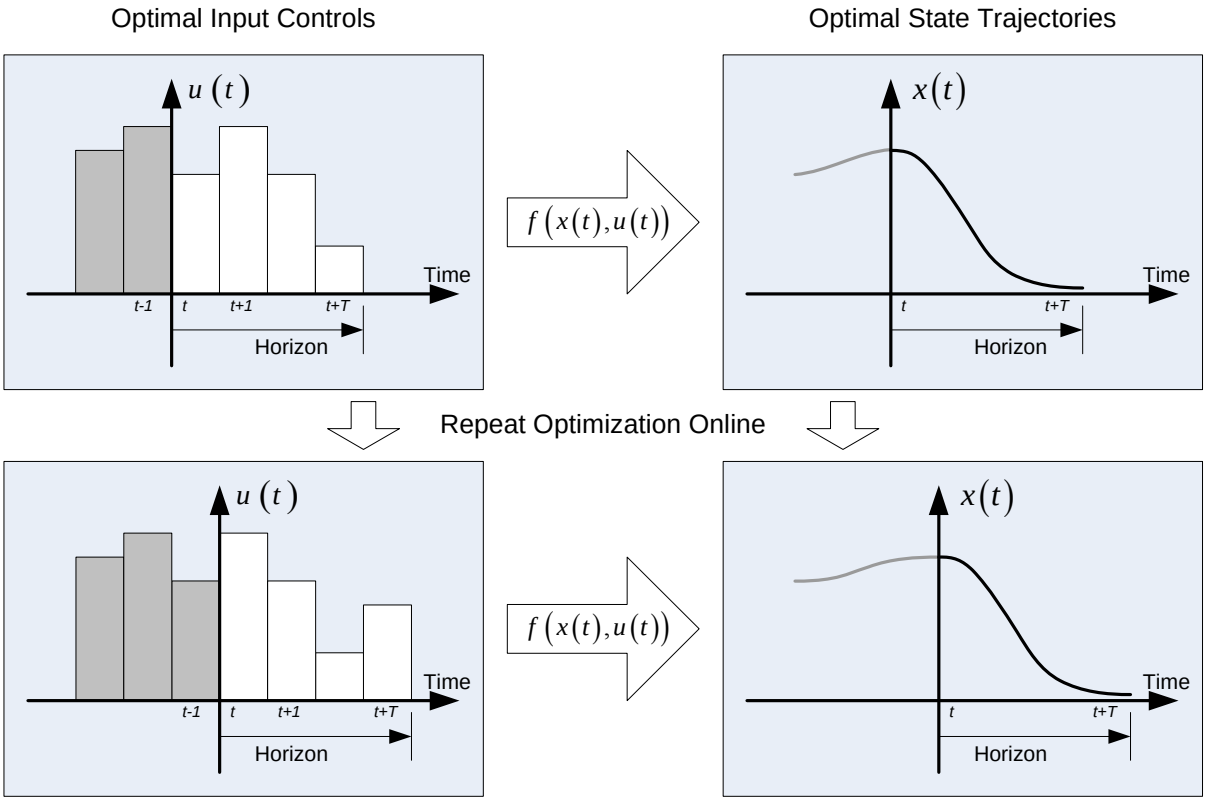

Figure 3-1: Receding horizon control is the optimization of an open-loop sequence of system inputs, in order to produce a minimal cost state trajectory and then repeating the process online. This diagram shows a typical piecewise constant input sequence with the corresponding measured and predicted state trajectories.

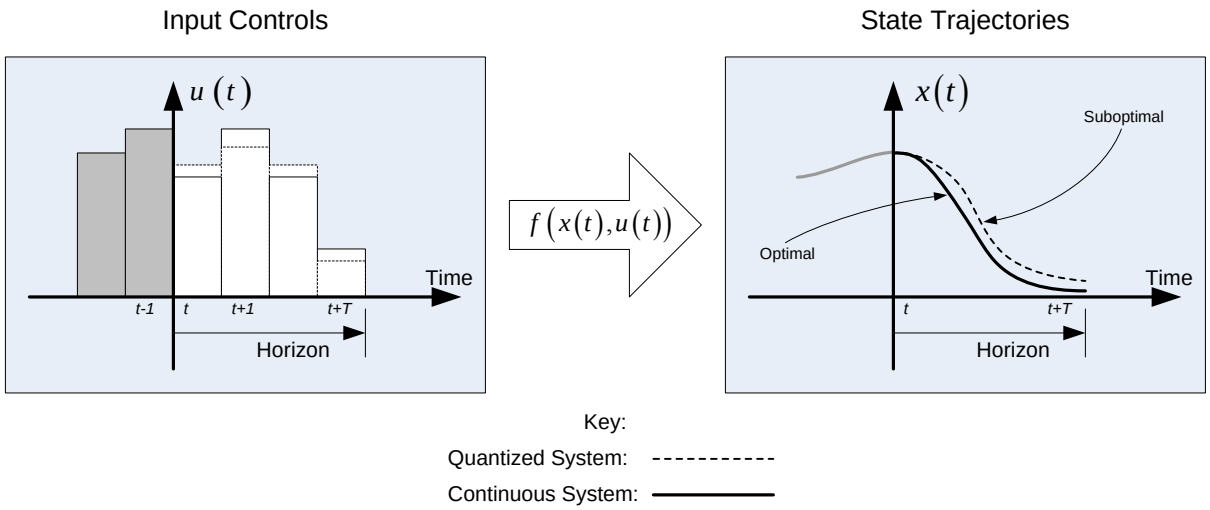

Figure 3-2: The quantization of the system's input space prevents the optimization problem from achieving true optimal performance. The true optimal control input lies somewhere in the control space, but the probability of finding that sequence in the quantized input space is very low. Hence, the quantized system will always achieve only a suboptimal state trajectory.

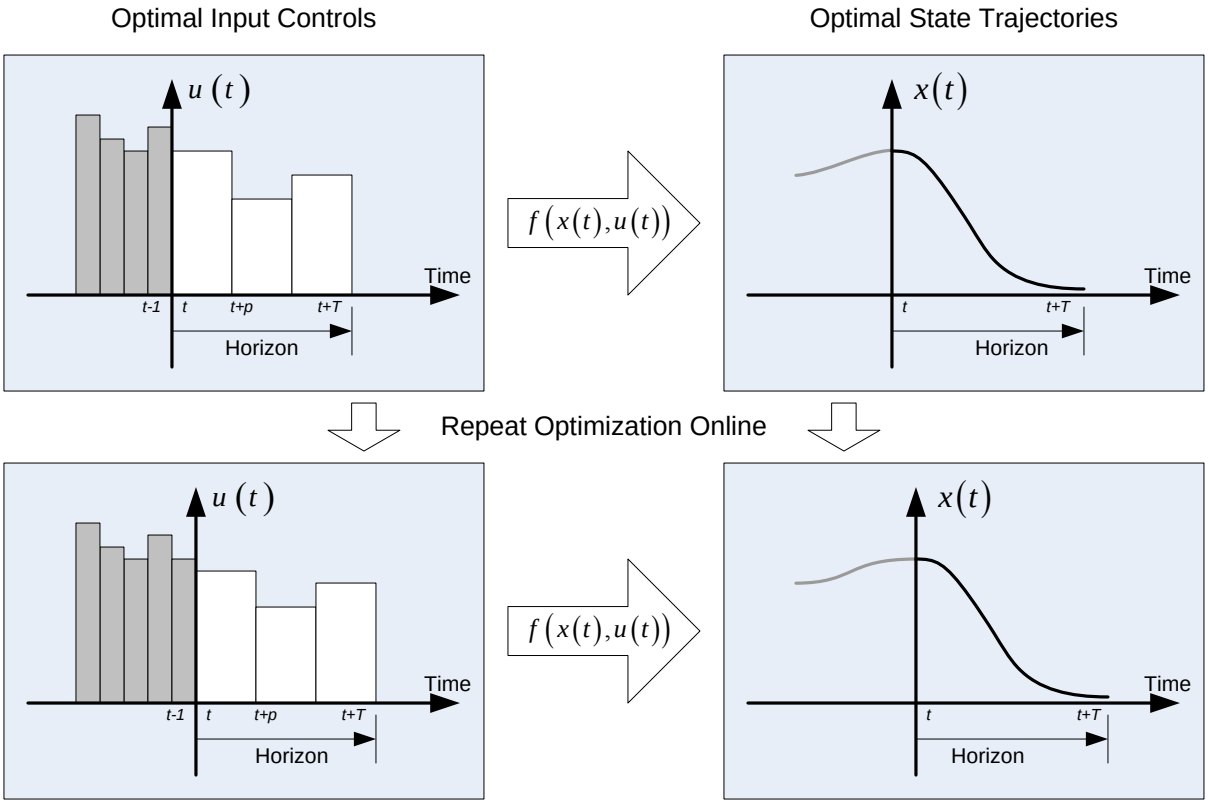

Figure 3-3: This diagram identifies the basic DFRHC scheme. Notice that there are two distinct time frequencies, one for state feedback and re-optimization, and the other for the prediction steps to the horizon. The process optimizes an input sequence, predicted with some input time period p, and then repeats the optimization after state information has been updated after a single feedback period.

CHAPTER 4
APPLIED APPROACH AND IMPLEMENTATION

Introduction

The need to implement online planning and real-time control on the CIMAR NaviGator AGV was the driving motivation behind the work presented in this dissertation. Using the theoretical concepts and mathematics described in Chapter 3, simultaneous planning and control was put into operation on the vehicle through the use of Heuristic Receding Horizon Control (HRHC), combined with Dual-Frequency Receding Horizon Control (DFRHC). Both of which are the new and novel theoretical concepts presented by this dissertation.

The NaviGator has served, as the basis platform for implementation and testing of these concepts. In the overall NaviGator control system, the implemented novel planning and control task is compartmentalized into a software component called the Reactive Driver (RD). Figure 4-1 shows a block diagram of this control system with the RD component residing within the Intelligence element. The RD allows the planning and control functionality to be decomposed into a separate compact module. By breaking the overall system down into smaller components such as this, the implementation of other tasks and capabilities like environment sensing, and low level vehicle actuation, becomes less complicated because they are only loosely coupled to the rest of the system.

The RD is responsible for closing the loop between sensors and actuators. As such information sensed from the environment is first processed by feedback components called Smart Sensors. These sensors process information from sensed raw data and normalize it into a standard form which can be easily fused with similar data from other Smart Sensors. The exact format of and information represented by this data will be discussed in the second section of this chapter. After the sensor data is fused together by an Arbiter component, it proceeds to the

Reactive Driver, where the RHC algorithm of this dissertation attempts to find actuator commands which regulate the vehicle onto a predetermined motion path. This flow of information is inline with the sense-plan-act scheme mentioned in chapter I, since it is repeated continuously throughout the systems online operation, and also because the planning step is inherent to the RHC functionality of the Reactive Driver.

The use of environmental sensors within the system leads to one major issue which is not theoretically addressed in the previous chapter. Namely the issue that comes into play during implementation is Obstacle Avoidance (OA). The introduction of obstacles into the vehicle's task and configuration space has severe repercussions on the system's stability and performance. For RHC to work, it is essential that the OA problem be cast into a form that allows the optimization procedure to find a valid control solution that will guide the system safely through the state space. This requires the obstacles and environment to be represented in a way that elegantly blends the OA problem with the trajectory or state regulation problem during the online computation. In turn, the value function must be modified to reflect the inclusion of additional environmental or state-environment interaction costs.

An essential section of this chapter addresses in detail the issues of OA. It is a vital discussion because OA capability for AGVs is what allows them to operate and execute different missions through their often cluttered and unpredictable environments. Since knowledge of the environment structure is usually incomplete prior to task execution, the robotic vehicle must have some OA functionality. This also means that the system must also have the ability to sense any obstacles or other undesired states, and a way to represent what is sensed so that it can be used by other processes that plan the vehicle's motion. This ability, on the NaviGator vehicle, is implemented as mentioned, within the Smart Sensor components, and is not the focus of this

dissertation. However, the output of those sensors is consumed by the Reactive Driver component of the system and therefore needs to and will be discussed.

Aside from the ability to sense disagreeable state configurations, an AGV can also benefit from the capacity to sense desirable states, terrain or conditions. This allows the system to maximize the chances of successful task execution. An example of this would be sensing a road adjacent to the vehicle's current path and then switching onto driving that road if it were more conducive than the current path. This ability, like OA, can also be incorporated into the RHC task, assuming that the desirable state configurations are represented in a way that allows such information to be combined with state regulation costs, within the optimization routine. The RD component in conjunction with the NaviGator's Smart Sensors, allows for this behavior, and is implemented in unison with the OA behavior. The ability to both avoid obstacles and track desirable terrain is inherent in the design of the Smart Sensor / Reactive Driver interaction, and will be addressed in detail later in this chapter.

Obstacle Avoidance

From Remark 1 in the previous chapter, it is suggested that RHC applied to an autonomous vehicle system will allow for unified motion planning and control. If the optimization problem is cast correctly, this should inherently include an OA capability. However, it should first be considered that the regulation of a vehicle onto some desired state motion structure, while at the same time avoiding any undesired regions in the configuration space (due to obstacles, or other high risk hazards) is by nature contradictory. For example, if the system is required to drive along a particular geometric path, avoiding an obstacle in the way of that path also demands that the vehicle depart from the initial intended trajectory. This behavior, when viewed from the classical control sense, is unstable with respect to the goal. Error signals used for path regulation would inherently increase during the avoidance maneuver. However, the overall behavior, when

viewed from the system user, seems desirable and in a new sense "stable." The implications of this simple concept are far reaching in their impact on the notions of system stability and performance. To simply put that the idea of stability requires the continuous minimization of all error signals for all time is no longer valid. The desired performance of the system, and what could be considered "stable behavior," now involves the movement around and away from the original goal structure.

The essence of these concepts is that stability, in the presence of obstacles, requires the minimization of a properly designed value function, not the minimization of only state error signals. This therefore suggests that unstable behavior is coupled to the sub-optimality of the cost function and open-loop input sequence of the RHC process. Meaning that if the value function, which incorporates a cost component due to environment, and obstacles, is not minimized then the probability of unstable behavior such as obstacle collisions increases.

A value function designed to provide both stable control (motion regulation) and stable behavior (obstacle avoidance), must incorporate both classic state regulation costs, and a state environment cost. The state environment cost adds a new term(s) to the value function, which was not discussed in Chapter 3. This new component describes the cost associated with the system occupying a given state in it operating environment, which in general can be attributed to obstacles, terrain quality, etc. Equation (4.1) presents a new cost function form with an additional term $P(x)$, which describes the added cost due to the system's state within its environment. It effectively trades off state error signal value with obstacle avoidance value. Meaning that if the cost of occupying a given state due to $P(x)$ increases, the overall optimization will allow for a slight increase in the remaining two terms in order to reduce the cost due to the environment. The remaining terms are identical to the value functions discussed in the previous chapter. The cost

due to error in motion regulation is expressed within $Q(x)$, and the cost due to input command is given by $R(u)$. Notice that both $P(x)$ and $Q(x)$ are both functions of the systems state. The difference in these two functions is that, $Q(x)$ is a mapping of state to error signals (which are based upon a predetermined motion structure) and $P(x)$ requires a mapping of system state to environmental cost.

$$V(x) = \sum_{u} P(x) + Q(x) + R(u) \tag{4.1}$$

The addition of the new term $P(x)$ to the RHC value function has a significant impact on the Feasibility Theorem for Suboptimal RHC referenced in the second section of Chapter 3. Of the three conditions of that theorem, only one can be partially satisfied when the state environmental cost $P(x)$ is included.

The first condition states that the RHC value function must be lower bounded by some class *K-function* $\alpha(\cdot)$. Assuming the term $P(x)$ is designed such that it is always positive, this condition can still be satisfied, since it is simply adding a positive number to the remaining portions of the function which can already be lower bounded by $\alpha(\cdot)$. However, the condition also states that the value function is equal to zero at its origin. Since $P(x)$, must be allowed to have a value greater than zero for cases where obstacles are present at the origin, this condition cannot be fully satisfied.

The second condition states that the cost function V(x) must be monotonically decreasing for successive time steps of the RHC solution sequence. This criterion cannot be satisfied when obstacle avoidance is included in the optimization problem. The reason for this is that in order to

avoid an undesirable state during motion tracking, additional cost may be incurred on by the value function in order to produce a desirable avoidance maneuver.

Finally the third condition cannot be guaranteed since the uniqueness of the control input for any given state can no longer be certain.

A simple example that demonstrates all of these cases is depicted in Figure 4-2, which shows a vehicle attempting to track a straight line and a single obstacle in way of that track. In this case it is possible to find two opposite but equal solutions to the RHC problem. Therefore, it is not possible to guarantee the uniqueness of a control solution or to ensure the control input is a well behaved function of the system's state, when obstacle or environment cost is included in the RHC value function. Also, since the cost of the value function inherently increases during some OA maneuvers such as the one depicted in Figure 4-2, it is not possible to satisfy the decreasing cost constraint which is required to prove feasible stability.

Although it can be shown that cases may exist which disagree with the criteria presented by the Suboptimal Feasibility Theorem, under nominal conditions they may still be satisfied whilst including obstacle avoidance as part of the RHC optimization. In addition, for all cases in which the state environment cost function $P(x)$ is zero, such as when no obstacles are present in the immediate environment, then the value function simply reduces to the classic RHC form, and all of the stability criteria may be applied as usual. However, for simplicity in this implementation, the three criteria are not explicitly verified. Rather, the control solution is guaranteed to be admissible, by designing a valid A* heuristic function. This topic is discussed in detail in the next section.

The ability to formulate the environmental cost component $P(x)$ online, allows the RHC optimization problem to dynamically modify the vehicle's predicted motion, in order to avoid

obstacles or drive on more desirable terrain. This online cost mapping is a functionality that must be provided by exteroceptive sensors since the required information is originating from the vehicle's surroundings. Thus information provided by those sensors must be in a form that effectively represents the cost of occupying a given state in the workspace. For implementation on the NaviGator, this information is provided in a form known as a traversability Grid. The traversability grid models the system's workspace as a raster image in which each pixel contains a value that estimates how desirable the terrain and space contained within that pixel is for the vehicle to occupy. The traversability value is effectively a gain like parameter that allows for a tradeoff between input effort or motion error and quality of terrain. The worse the terrain, the more willing the RHC is to add error cost to the determined path in order to circumvent that poor area.

Obstacles are represented by a traversability grid in much the same way as they are in an Occupancy grid. Both positive and negative obstacles (such as pot holes) within the environment can be mapped equivalently to poor traversability regions. In fact, this is one feature of the traversability grid that cannot be effectively represented by an Occupancy grid. In an Occupancy grid, only information about the "occupied" or "free" state of a pixel is given, whereas a traversability grid can encapsulate these pieces of information and also describe terrain quality. This means that any type of terrain can be mapped into traversability space without having to convey anything specific about the given topography. In the traversability sense, positive obstacles appear the same as negative obstacles or steep slopes. Traversability also allows for the evaluation and scoring of unoccupied space. For example, pavement can be classified as more traversable than sand, grass or gravel. Therefore the vehicle is able to select the best possible path for motion when it is presented with different types of terrain.

The units of measurement for the values contained within the traversability grid must also be carefully considered. As aforementioned, the values are used to tradeoff occupying a given space with motion structure error and or input effort. As such, the grid values are essentially dimensionless, and do not describe an amount of travel energy required or probability of safe motion, etc. This is because values with these types of physical units would inherently change as a function of the system's complete state (i.e. speed and orientation), and would require much more information about the environment to be mapped by sensors. It would also require a more complex $P(x)$ cost function. Therefore, traversability as discussed in this dissertation is only considered as a qualitative parameter that can simply and effectively allow the Reactive Driver component to make decisions about the desired vehicle motion. Figure 4-3 presents the traversability grid concept. In this image a model of the surrounding local environment is mapped into the corresponding traversability space. The area outside of the vehicle's commanded operating space is all mapped to poor (shown in red) traversability values. The values go from shades of red, which are poor, to shades of green and blue, which are desirable.

In implementation on the NaviGator system, the traversability grid format for all sensors feeding the Reactive Driver component is a 121 by 121 pixel rasterized image with a resolution of 0.5m by 0.5m per pixel. Each pixel in the grid contains is a 4-bit integer, which can represent a traversability value or one of several special case values. The grid is designed such that the vehicle is always positioned at the center of the image, and as the vehicle moves the values within the image are shifted so that the vehicle remains in the center. When shifted, new values around the edge of the image are updated with new data from sensors. The coordinates within the image are easily mapped to local vehicle coordinates and therefore the traversability grid serves as the environmental cost function $P(x)$, for implementation in the RHC value function.

However, since the orientation of the vehicle at each pixel within the image is not considered, only the vehicle's position within the grid is used to determine environmental cost, and not its full state (position and orientation).

The complete value function used for implementation in the RD component, which includes traversability, cross track error, and heading error, has been designed as:

$$V(x(t)) = \sum_{\tau=t}^{t+T} \left(k_{trav} * Trav\left(x(\tau)\right)^2 + k_{herr} * Herr\left(x(\tau)\right)^2 + XTrack\left(x(\tau)\right)^2 \right). \qquad (4.2)$$

Where the sub function $Trav\left(x(\tau)\right)$ accounts for the environmental cost due to obstacles and terrain quality, and has a corresponding optimization weight k_{trav}. The remaining two sub functions provide transformations of the system's state into path tracking error coordinates, and allow for optimization of the vehicles motion tracking of a desired trajectory. The heading error function, $Herr\left(x(\tau)\right)$ is multiplied by an optimization weight k_{herr}, whereas the cross track error function, $XTrack\left(x(\tau)\right)$ is not altered. This allows for normalization of cost units, using the cross track error as the reference base for overall value. Figure 4-4 presents a geometric representation of the tracking error system measurements, Herr and XTrack.

Thus the implementer is free to modify the remaining weight factors to tradeoff traversability, and heading error with cross track error as a baseline. Note that input effort / control value has been neglected here since there is not term in the function penalizing it. It has been found through extensive testing that desirable results are obtainable without control effort in the cost function, and that the overall optimization is less constrained to find an appropriate solution with the value function provided in equation (4.2).

For implementation of HRHC where the A* search is used in place of a classic optimization routine, the depth cost function $G(n)$, must be specified. Since the general value function provided in (4.2) specifies the cost structure for the underlying problem, a variation of that function is used as the depth function:

$$\begin{aligned} G(n_i) = & k_{trav} * Trav\left(xnode(n_i)\right)^2 \\ & + k_{herr} * Herr\left(xnode(n_i)\right)^2 \\ & + XTrack\left(xnode(n_i)\right)^2 \\ & + G\left(pnode(n_i)\right) \end{aligned} \tag{4.3}$$

$$\forall pnode(n_i) \neq \varnothing .$$

In this form the depth function is defined recursively, and is dependent upon the depth cost of the parent of node n_i. For the root node (s) the depth function is simply:

$$G(s) = 0 . \tag{4.4}$$

Admissible Heuristics for HRHC

The cost function presented in the previous section provides a means for the RHC implementation to optimize both obstacle avoidance and motion tracking online. In the previous chapter the new technique, Heuristic Receding Horizon Control (HRHC), offers a means to use heuristic information and the A* search algorithm to implement the optimization task within the overall RHC scheme. One essential requirement for HRHC then is to establish a heuristic function which can estimate a cost-to-go for the value function provided in the second section of this chapter. As will be shown in this section the heuristic function can take on many different forms, each with various strengths and weaknesses. Finally the form decided upon for implementation in the NaviGator's RD component will be presented.

Cost functions for a typical RHC and a typical A* implementation tend to be very different in nature. Classically, A* state to state transition or "arc" cost values are relatively constant or simply depend on the transition distance, whereas RHC state transition costs decrease as the system approaches the origin or goal. Regardless of this difference, the general method or technique for HRHC allows for a total cost function f (n) to take on any form, so long as it satisfies some basic criteria. First, the arc or state transition cost must always be greater than zero. Since the value function provided in (4.2), is positive definite and provides this cost measurement, the first condition is satisfied. The second condition requires that the heuristic function $h(x)$ be an underestimate of the total cost-to-go for the provided value function. Therefore, careful consideration for the heuristic function must be taken in its design to ensure that it is upper bounded by the true optimal total to the goal. If this condition is violated then the algorithm A* is no longer admissible and may determine an invalid solution sequence.

A good approach to heuristic function design is to analyze each term in the RHC value function, and to establish a heuristic estimator for each of those terms. Then all of the partial heuristics can be summed to obtain the complete heuristic function.

For implementation within the RD component on the NaviGator, it is desired to find a heuristic estimator that approaches the true minimum cost solution over the time horizon for equation (4.2). In analyzing that function term by term, it can be seen that the first term provides the cost due to environment, which is directly related to the mapped traversability value. The ideal heuristic for this term would provide a cost sum over the remaining time interval with the true traversability encountered along the optimal state trajectory. Since that trajectory is not known, an estimate or prediction of the encountered traversability must be made. Also, to adhere to the admissibility constraint of the A* search, the traversability estimate must not exceed the

optimal solutions value, so a conservative approach must be taken. In other words, the predicted traversability cost must be less than the actual future remaining time traversability cost. One simple heuristic function that estimates the future incurred traversability cost, at a given time (τ), within the HRHC search is:

$$\hat{h}_{trav}(\tau) = \left(\frac{t+T-\tau}{\Delta\tau}\right) k_{trav} \min_{\forall x} Trav(x)^2, \tag{4.5}$$

where $\Delta\tau$ is the time step period of the search. This heuristic estimates that the total future cost due to traversability is equal to the minimum traversability value of any possible state, multiplied by the traversability cost weight, and the number of future time steps. This function is the heuristic estimate used by the RD component in implementation. It provides a fast method for calculating an estimate which is guaranteed to be upper bound by the true future traversability cost. It is fast because the minimum traversability need only be calculated once, and then that minimum value may be applied for calculating each search node's heuristic cost. Other heuristics which are more informed about the underlying system dynamics and problem are able to calculate a more accurate heuristic. However, these functions tend to be much more demanding since the future minimum traversability is truly a function of the systems state and must be recalculated for each search node. Thus the time saved in reducing the total number of explored nodes, by using a better heuristic, is lost since the heuristic itself takes much more computation time, and so more advanced heuristics are not worth implementation.

Heuristic estimates for the heading error and cross track error cost components are both calculated directly from the system's state at any given node within the A* search. Unlike the traversability estimate which must have knowledge of the systems surroundings, and therefore must perform a separate search to find the future minimum traversability, the system state only

dependent costs of heading and cross track error may be calculated by pure geometric and kinematic constraints.

For the heading error heuristic design, the total incurred cost may be lower bounded by knowledge of the vehicle's minimum turning radius constraint, and a constant speed assumption. In more detail, since the NaviGator is a front wheel steered vehicle, there is a minimum turning radius that the platform is able to drive. At a given speed (the one at which the HRHC search is attempting to optimize motion) there exists a maximum turning rate for the vehicle. By calculating this maximum turning rate at any given node's state in the search, the future incurred heading error cost may be estimated with the equation:

$$\hat{h}_{herr}\left(x(\tau)\right) = k_{herr} * \sum_{\gamma=\tau+\Delta\tau}^{T}\left(\left|Herr\left(x(\tau)\right)\right| - \frac{v}{r_{\min}}\gamma\right)^2 . \qquad (4.6)$$
$$\forall \gamma < \frac{r_{\min}\left|Herr\left(x(\tau)\right)\right|}{v}$$

The above equation assumes that the heading error will decrease linearly at a rate equal to the maximum vehicle turning rate, where v is the vehicle's desired speed, and $r_{\min}$ is the minimum turning radius. The estimated future cost is therefore at least the summation of the minimum possible heading errors, up until the point at which a heading error of zero is possible, and from thereon it is assumed to remain zero. For all time greater than the time at which the heading error would be equal to zero, the future incurred cost is assumed to also be zero or

$$\hat{h}_{herr}\left(x(\tau)\right) = 0$$
$$\forall \left|Herr\left(x(\tau)\right)\right| < \frac{v}{r_{\min}}\Delta\tau \text{ ,} \qquad (4.7)$$

because it is assumed that the heading error will become zero and remain zero before the next search time at $\tau + \Delta\tau$, and therefore no future cost will exist.

The cross track error heuristic is calculated in much the same way as the heading heuristic. For this estimate, it is assumed that the vehicle's cross track position will decrease as if it were headed perpendicularly towards the desired motion path. This clearly will lower bound the true future cost since the vehicle must approach the path at an angle near parallel with a line tangent to the path, if it is to converge onto it. To calculate the heuristic then, a summation of the minimum future possible cross track errors is made. The following equation makes this calculation, within the RD component, and assumes that the vehicle's desired speed will remain constant

$$\hat{h}_{xtrack}\left(x(\tau)\right) = \sum_{\gamma=\tau+\Delta\tau}^{T} \left(\left|XTrack\left(x(\tau)\right)\right| - v\gamma\right)^2 \quad \forall \gamma < \frac{\left|XTrack\left(x(\tau)\right)\right|}{v} \tag{4.8}$$

Since the actual accumulated cost given in the value function (4.2) does not multiply the cross track cost by a weighting factor, then neither does the heuristic given in (4.8). As in the heading error heuristic it is also assumed for the cross track error, that once the error reaches zero, it will remain zero for all future time. Therefore for the heuristic estimate, the summation of future estimated cross track errors is only defined for those with positive values, and for the cases in which the future cross track error is zero the heuristic is also zero and is given by:

$$\hat{h}_{xterr}\left(x(\tau)\right) = 0 \quad \forall \left|XTrack\left(x(\tau)\right)\right| < v * \Delta\tau \tag{4.9}$$

The total heuristic function is simply the summation of the partial heuristics given above and can be expressed as:

$$\hat{h}\left(x(\tau)\right) = \hat{h}_{trav}\left(x(\tau)\right) + \hat{h}_{herr}\left(x(\tau)\right) + \hat{h}_{xtrack}\left(x(\tau)\right). \tag{4.10}$$

The above heuristic function provides a lower bound estimate of the true future incurred cost of the optimal state trajectory, and therefore should allow for the A* search to yield an admissible solution. The solution obtained should satisfy the basic feasibility criteria, necessary for stable Receding Horizon Control, especially under obstacle free circumstances. It should be noted that there seems to be a strong connection between the heuristic criteria of A* and the second feasibility implies stability criterion. Essentially if the heuristic function is not admissible then it may not always underestimate the true future cost to goal. In that case, the solution will not be admissible and will almost certainly violate the second stability criterion. This simple connection implies that an admissible heuristic function may account for the majority of the HRHC algorithm's stability and control effectiveness.

Reactive Driver Implementation

The Reactive Driver (RD) component of the NaviGator's overall control system houses the HRHC implementation of this thesis. Along with the cost functions and heuristics detailed in the previous two sections, there are a number of important functions of the RD that merit discussion, and complete the system integration puzzle, which allows for the RD to provide true functionality in the real-world system.

As shown in Figure 4-1, the RD is one of many components which makeup the complete autonomous control system. Each of these components is designed as per the Joint Architecture for Unmanned Systems (JAUS), and each houses a core functionality that is defined by JAUS. JAUS is a standard which specifies a basic structure and methodology for unmanned and autonomous systems design and integration. It is intended to aid in the integration of and to support interoperability amongst many heterogeneous unmanned systems originating from different developers. In addition to the core component architecture, JAUS primarily offers a means for common messaging and information exchange among its intended hierarchical

network of systems, subsystems and computing nodes. This messaging system is what allows the RD to communicate with and control surrounding components in the NaviGator's control system.

Like all components implemented on the NaviGator, the RD extends a core JAUS component template and a corresponding JAUS library, which are implemented in the C programming language. The software is built upon the Linux and GNU tool-chain, with its extensive set of libraries and compilation utilities such as the GNU C Compiler (gcc). This base understructure provides each of the components operating on the NaviGator a means to support timing processes, inter and intra nodal communication, as well as multithreading capability.

The RD component has been implemented to run on a computing node which is made up of a Gigabyte motherboard with AMD CPU, 1 GB of RAM, and a 4 GB Compact Flash solid state hard drive. The computing node runs the Linux Fedora Core 3 operating system, which allows for advanced software development as well as execution directly on the NaviGator. Also, the node is connected via a high speed Netgear Ethernet switch to seven other identical computers onboard the NaviGator. Each one running its own set of tasks and processes for the vehicle's control system, and all intercommunicating via JAUS base datagram packets which are routed through the onboard Ethernet switch.

The JAUS messaging communication provides essential data to the RD. Specifically, there are five data streams originating from other components in the control system, each of which the RD requires for complete operation. The data included are: the vehicle's global position and orientation, its velocity state, the state of the surrounding local environment (in the form of a traversability grid), feedback from low level actuators, and their health status. Each of these data streams is in a form known as a JAUS service connection, which allows for JAUS message

reports to be sent on a periodic basis without needing to re-query for the report every time it is desired.

Both the position and velocity JAUS messages originate from the same computing node which runs multiple components. One component is the Global Position and Orientation Sensor (GPOS), the other is the Velocity State Sensor (VSS). Both of these tasks process feedback from low level vehicle state sensors, such as accelerometers, GPS, and odometers, and cast it into the proper JAUS message for consumption within the rest of the computing network.

The traversability grid data is wrapped into a JAUS message which originates from the Smart Arbiter (SARB) component, and is a compilation of a number of other grids from lower level Smart Sensor components. Each of the Smart Sensor components maps data from a physical environment sensor, such as a camera, to a single traversability grid. Then each grid is sent to the SARB where it is fused together, to form a composite and more complete estimate of the local environment.

Lastly, information from the component directly below the RD in control loop, the Primitive Driver (PD), is fed back up to the RD level. This information includes two data streams. One stream contains information about the low level vehicle actuators (steering, throttle and brake), specifically the current position of each actuator, and is in the form of a JAUS report wrench effort message. (A wrench describes a set of forces and moments, and supports the way JAUS abstracts low level platform control.) This wrench is needed to keep an updated model of the vehicle, mainly for determining the current front wheel steering angle, and thus allowing for accurate state predictions to be made during the HRHC routine. The second stream contains information about the health status of the PD itself. This data is important to the RD in order for it to maintain control of the vehicle, and to take evasive action if there is a fault within the PD.

All JAUS components on the NaviGator are implemented as finite state machines. Each one can assume one of seven possible accepted states; they are enumerated and named: Startup, Initialize, Standby, Ready, Emergency, Failure, and Shutdown. By monitoring the state of the PD the RD is able to infer its health status, and therefore able to maintain a more fault tolerant level of control. In most cases the component executes its main task while it is in the Ready state. The other states such as Initialize and Standby are included to allow the component to switch its behavior in order to establish needed data service connections, disconnect from the rest of the system, or to pause for operator control.

The RD Ready state software is mainly composed of one control loop. Within this loop all of the necessary procedures and steps are taken in order to execute its primary purpose which is to command the desired vehicle control wrench to the PD. This wrench is the only output signal / data stream provided by the RD, and its purpose is to command the vehicle's steering, throttle and brake actuators such that the intended motion of the Receding Horizon Controller is executed. The steps in between updating local software variables from the input data streams, and the output of the command wrench, include fault detection, determination of the vehicle goal state, traversability grid modification, and finally the execution of the HRHC and a simple PID loop to control the vehicle's speed.

Table, outlines each of these important steps and will be used for reference within this section and the rest of the chapter.

In the first step of the RD control loop data is updated from the five incoming service connections detailed above. The next step in the procedure begins to use that data in order to detect if any possible faults have occurred. Specifically, in verifying that all critical communication streams are still active and that the PD component is still in a health state,

waiting to be commanded. The third step will simply switch the RD into the Standby state if it is so commanded by the operator. This command is determined by checking the state of the PD, and observing if it is also in the Standby state, if so then the RD follows into Standby.

The fourth step in the procedure requires some special consideration. This is the first step in the procedure which requires data from the input path file. The path is provided to the RD by an operator as a flat data file, and specifies an *a priori* motion plan for the vehicle. The file contains a data structure which is made up of a list of piecewise continuous path segments. Each segment is specified with a starting point latitude and longitude, end point latitude and longitude, desired speed, and segment curvature. At this step the RD determines which segment the vehicle should attempt to track by analyzing the vehicle's current position with respect to the path segment ahead of the segment it is currently tracking. If the vehicle is closer to the next segment than it is to the current segment, then it will switch to tracking the next one in the sequence.

Since the desired speed may vary from segment to segment, it is necessary in the following step to update the current desired speed of the vehicle. This step requires looking forward in time on the desired path to check for lower speeds and taking into consideration the vehicles deceleration capability. Lower speeds on the path might also be attributable to the curvature of the desired path segment. If the curvature is high enough, it may require the vehicle to slow down in order to track that segment without risk of rolling the vehicle. This procedure is done by finding the minimum path segment speed (due to desired or curvature value) over a nominal deceleration period, and then analyzing if that speed requires the vehicle to begin slowing down at the present time. If it does, then the current desired vehicle speed is set to a value which accounts for deceleration time. If it does not, then the current desired speed is simply set to the desired speed of the current path segment. This functionality is equivalent to the equation,

$$DesiredSpeed = \min_i \left[speed(s_i) + D * distTo(s_i) \right]. \quad (4.11)$$

Where the functions $speed(s)$, and $distTo(s)$, determine the desired speed of path segment s, and the distance from the vehicle to the start of the path segment respectively. The constant D accounts for a nominal distance based deceleration, measured in meters per second per meter. The value used on the NaviGator is tuned to: $D = 0.25 mps/m$.

The fifth step, of the RD control loop calculates the latitude, longitude and size of the desired RHC goal region. The region is a circle centered at the determined coordinates and its size is simply determined by a radius value, calculated in units of meters. The center of the area is determined by using the RHC planning time horizon period, and the a priori path data. The algorithm projects the vehicle's current location onto a point perpendicular to the path, and then extrapolates out in time, using the current desired speed, to a point that lies somewhere on the desired path ahead. The radius of the goal region is scaled linearly with the desired vehicle speed, to allow the discrete planner enough space to seek out, rather than a fixed size goal, which might be too narrow to find.

Two special cases exist in determining the goal point. The first occurs when the goal is found to lie outside of the locally mapped traversability space. Since the planner requires knowledge of the traversability value at any given state, points outside of the grid region are undefined and therefore seeking a goal in that space would require some significant assumptions. Rather than plan outside of the traversability defined region, the goal point is simply projected onto its boundary, where it meets the desired path. The second special case occurs when the end of the path is encountered within the time horizon. If so, the goal point is just set to the endpoint of the last path segment.

With the desired speed and goal point known, there is one last step required in implementation before executing the HRHC algorithm. The step is called grid dilation and serves a very practical purpose for obstacle avoidance. Since the vehicle takes up a region of area within its local environment, it is necessary to account for its size during planning, in order to ensure that there is enough clearance around the vehicle to completely avoid any obstacles. One method to do this would be to find the minimum traversability value within a region representing the vehicle's footprint, in order to calculate the traversability cost at any point in the planning search. This would thus require a separate minimum traversability value search for every node expanded during the A* algorithm's execution. Since there are many nodes expanded during a nominal search, and many pixels to explore for any given footprint configuration, this operation would be very cumbersome. A more efficient method is to expand any obstacles within the traversability grid, by the approximate size and shape of the vehicle, and then assume the vehicle occupies only is a single point in space, during the search routine. This dilation then need only be done once per iteration, rather than many times during the search itself.

Also, since the traversability value is a general description of environment cost, and not purely an obstacle representation, it makes sense to dilate the grid by the any minimum traversability value within the expansion area, and not solely obstacle values. This guarantees that the search will account for vehicle size and shape, even when seeking out desirable traversability regions, or avoiding regions of intermediate cost.

The dilation procedure is implemented as a pixel by pixel loop over the entire grid. For each pixel the dilation value is determined by looping again through an area around that pixel, and finding the minimum traversability value within that area. The search area is called the kernel, and it represents the approximate size and shape of the vehicle. Since the orientation of

the vehicle at any given node within the search may vary, the shape of the vehicle, which is ideally rectangular, cannot be assumed in any one configuration. Therefore, the shape of the kernel is assumed to be a circle, with a radius slightly larger than the half width of the NaviGator. Figure 4- is an example of what a traversability grid image looks like before and after circular dilation. The circular shape guarantees that if an area is avoided in the dilated image, then the vehicle should at least have enough clearance to pass by it at its side. Since the vehicle may be approaching the region at a slightly different orientation, the kernel radius is tuned to be slightly larger than what is probably needed in order to account for its difference in shape.

At this point in the RD control loop, all of the required data and values have been determined or modified such that the Heuristic Receding Horizon Controller (HRHC) can be executed. The purpose of this step is to determine a control input that when commanded to the system will yield the desirable motion to both maintain path tracking and if necessary avoid any obstacles. The form of this control input is the steering wrench effort portion of the JAUS wrench message that will be sent to the Primitive Driver for low level vehicle actuation. This value varies from -100 to 100% effort, and essentially maps directly to the steering system for turning the front wheels anywhere from 100% left, through 100% right.

As discussed in the fifth section of Chapter 3 of this dissertation, the HRHC algorithm is implemented as a modified A* search. The first step in the routine creates the root node by using the current vehicle state feedback (position and orientation), and the current control (steering effort). All nodes in the search are stored as a C language data structure, with variable members including: cost-to, cost-to-goal estimate, parent node pointer, generation number, and vehicle state / control information. A fixed number of theses nodes are allocated in memory prior to running the routine for the first time. They are stored in a linked list data structure, named in this

implementation “Node Dispenser”, which allows nodes to be divvied out as needed, and can be reset quickly by simply resetting a single pointer that indicates the node to be dispensed next. This Node Dispenser method has been implemented to optimize the code and increase efficiency, verses allocating and freeing a single node, every time one is used.

The next step in the algorithm establishes the root node onto a set of nodes called the Open Set. In this implementation, the Open set has been optimized for searching and sorting according to the cost-to-goal estimate which represents the A* $\hat{f}(n)$ value. The optimization is done by casting the Open set into a data structure known as a heap stack. A heap stack is a stack of nodes divided into multiple levels. The first level in the stack contains a single node, the second level contains two nodes of which the node above is parent to. Each level after can store twice as many values as the one above it, because the two child per parent relationship continues on as the stack grows. In this manner, the stack expands exponentially in the horizontal direction as it grows vertically. The heap stack is design to maintain one special property, that is, the parent node value is guaranteed to be less than (or greater than, depending upon if the stack is desired to be ascending or descending) the two child node values. This relationship ensures that the value at the top of the heap stack will always be the minimum or maximum of all nodes. This is very desirable for the A* implementation, since finding the minimum cost-to-goal estimate node on the Open set is a crucial part of the algorithm. Having the Open set organized in this fashion allows for that node to be found in a single step, no search need be conducted. The heap stack however, requires to be reordered when a node is popped from or pushed onto it, but because of the exponentially growing nature of the heap, this reordering is efficient and can be done in $O(\log n)$ computation time, where n is the number of nodes on the stack.

The ability to quickly find and remove the minimum cost estimate node from the heap stack is taken advantage of in the next step of the algorithm, where it is removed from the Open set and then checked for membership in the goal set. This is done by analyzing if the node's vehicle state is in the goal region. If it is, then a solution node has been found, if not then the A* search continues by expanding a new set of child nodes through the expansion function $\Gamma(n)$.

The expansion function for HRHC is implemented by generating a new set of possible control inputs and then predicting their effect by propagating the current node's state through a vehicle model function. The manner in which the control inputs are generated is defined by the artificial input space quantization that has been discussed in fourth section of Chapter 3. The true input that can be commanded to the PD is a continuous value in the range of -100 to 100% effort. However, this would require generating far too many nodes to be practical. Therefore, a finite number of nodes, representing a finite set of quantized input commands are expanded, and the one sequence yielding the least cost trajectory is found. The first input command associated with this sequence is then delivered as the control.

For the RD, the input space quantization resolution is a tunable parameter that can be increased or decreased in order to improve optimality, or increase speed. This means that as the quantization resolution is increased, the A* search is able to find a more optimal solution, whereas if the resolution is low, the solution is less optimal, and there is a chance that may not be found at all. Also the finer the quantization, the more elegant the vehicle control, because the input commands will tend to vary only slightly between control iterations. If the quantization is low, then as the commands change between control iterations, the vehicle will tend to nudge and jolt as the steering wheels move back and forth between commands.

The input commands for the NaviGator are generated ad hoc to take advantage of one key vehicle constraint. The steering wheel actuator on the NaviGator has a rate limit which only allows the wheels to turn up to a maximum speed. This rate limit corresponds to a maximum and minimum input command at each execution of the node expansion function. Since the steering is rate limited, and there is a finite time between commands in the horizon search, then any command higher than the rate limited maximum would yield the same result as the maximum itself. Therefore, expanding any nodes with input commands higher than the rate limited maximum will not add any value to the search and will only waste search effort. This constraint thus allows the input space to be truncated on an expansion basis, and so resolution is increased in the truncated space when the number of divisions is held constant. The rate limit on the NaviGator vehicle was measured to be: $60\ effort\% / \sec$.

Also, in an effort to minimize search complexity, the quantization resolution is decreased as a function of the search depth. This is done because the search resolution as the depth increases is less and less critical to successful control, because the goal of the search is to find the first input command which leads to a successful state trajectory. Therefore, higher control resolution is most important at the root node expansion.

For each control input determined during the quantization procedure, a search node is generated and then its vehicle state is copied from its parent node. This state is then is extrapolated through the vehicle model function over the horizon planning time interval, using the newly determined control input. The vehicle model used is kinematics based, and works by projecting the vehicle's position and orientation along a straight line, or circular arc, depending upon the steering curvature of the vehicle state. Since the rate limit of the steering system has a significant impact on the NaviGator's maneuverability, it is also taken into account in the vehicle

model. The model function first determines how much the wheels will move in the given time step, based upon the input command and the rate limit. The effective path curvature is then calculated by averaging the steering path curvature at the beginning of the time step with the steering curvature at the end of the time step. It is assumed that there is a linear relationship between the steering effort and the geometric curvature of the path that the vehicle will drive. The relationship can be equated as follows:

$$k_{path} = K_{effort} * SteeringEffort . \tag{4.12}$$

This is a simple mapping given the steering effort value from -100 to 100%, and the constant K_{effort}, determines the effective curvature of the path the vehicle will drive. As long as the front wheel steering angle remains significantly less than 90 degrees, then this assumption will be valid. On the NaviGator the mapping constant was measured to be:

$K_{effort} = 0.0016 / effort * m$.

Since the geometric curvature is simply the multiplicative inverse of path radius, the new vehicle position and orientation may be calculated once the curvature is known. For the model, it is assumed that the average curvature over the time interval is effectively what the vehicle will drive, because the change in curvature in each planning step tends to be small, this assumption is relatively benign. Also, if the average curvature is found to be very small, then it is assumed that the vehicle is simply driving along a straight line, and so the new position and orientation state is calculated as such. However, if the average curvature is outside of the straight assumption threshold, then the new state is extrapolated along a circular arc, with radius equal to the inverse of curvature. Both the straight line and circular projections assume that the vehicle speed is

constant over the time interval $\Delta\tau$. Here the equation for calculating the new state position along a straight line of motion is:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_{prev} \\ y_{prev} \end{bmatrix} + v * \Delta\tau * \begin{bmatrix} \sin(\theta) \\ \cos(\theta) \end{bmatrix} \quad \forall |k_{ave}| < k_{threshold} . \tag{4.13}$$

The state yaw angle θ is measured in global coordinates where true North is equal to zero, and East is 90 degrees. Therefore equation (4.13), calculates the new x position using the sin function and the new y position using the cos function.

The new position along a circular arc is calculated using a somewhat more lengthy equation. This is done when the average curvature exceeds the straight line approximation threshold and is given by,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_{prev} \\ y_{prev} \end{bmatrix} + \frac{1}{k_{ave}} \begin{bmatrix} \cos(\theta)(1-\cos(\Delta\theta)) + \sin(\theta)\sin(\Delta\theta) \\ \cos(\theta)\sin(\Delta\theta) - \sin(\theta)(1-\cos(\Delta\theta)) \end{bmatrix} \quad \forall |k_{ave}| > k_{threshold} . \tag{4.14}$$

For implementation in the NaviGator RD, a curvature threshold: $k_{threshold} = 0.01$ was found to work well.

During node expansion the traversability arc cost is calculated by analyzing the vehicle's state transition through the grid as it moves from its starting position and orientation to its end. This is done by measuring which values it encounters and then calculating an average over the interval. The grid values encountered during a state transition from one point to another are found by assuming the vehicle moves along a straight line from the start point to the end point. Since the grid resolution is somewhat low, 0.5 m by 0.5 m, this assumption is valid, because the pixel values touched by the state trajectory would vary little between a short circular arc path and a short straight line path, which is mostly the case. Bresenham's line algorithm [BRE63] is used

to determine which pixel values to measure along the state transition line. This algorithm is highly optimized for calculating the pixel coordinates of a straight line through a rasterized image space, and so it is very desirable for this implementation, where thousands of small lines need to be measured during each search.

One option the operator has when executing the RD component, is to plot the image coordinates in the traversability grid, which are measured during the HRHC algorithm. This allows for the effective search space to be analyzed at runtime. Figure shows the result of this plotting within a sample traversability grid on the left. As shown, the HRHC routine creates a tree like structure of possible state trajectory paths through the local environment. This structure corresponds to the quantized control inputs which are generated and extrapolated through the vehicle model. By analyzing this area searched by the algorithm it is possible to gauge the optimality of the solution trajectory. If the area searched both relatively large and dense, then the solution path should be the minimum of all paths within that area, and therefore has been weighed against many possibilities. However if the searched space is sparse, then the solution may be poor, because not enough other possible solutions have been checked to verify than nothing better exists. This is one of the key methods used in tuning the HRHC algorithm. Parameters such as the number of quantized input commands, horizon time, and planning time step, are tuned so that the search will be as broad as possible, while still remaining fast enough for real-time execution.

Finally, after a solution trajectory has been found, it is traced back from the goal node to the root node via the parent node pointers. The node directly before the root node is reached, contains the control input value that is then used as the steering command for the iteration of the control loop. At this point the HRHC algorithm is complete.

The next step in the overall process is to execute a simple PID controller to maintain the desired vehicle speed. This controller produces two mutually exclusive commands. One is for the throttle and the other for the brake actuator. They are found by executing the PID controller given the set point speed, and the current vehicle speed feedback, in order to determine a value called the linear control effort. If the effort is positive, then it is multiplied by a throttle gain value, which determines throttle command sent to the PD. Also if the linear effort is positive, the brake command is simply set to zero. Conversely, if the linear effort value is negative, then it is scaled by a brake gain, and then delivered to the brake, while the throttle value is held at zero. The general PID controller form is modified slightly with a feed-forward gain k_{ff}, and a constant bias value b_{effort}. This linear effort control value L_{effort} is calculated by:

$$L_{effort} = k_p e_v + k_i \int e_v dt + k_d \frac{de_v}{dt} + k_{ff} v_{desired} + b_{effort} . \tag{4.15}$$

Where k_p, k_i, k_d are the PID gains, e_v is the instantaneous velocity error, and $v_{desired}$ is the instantaneous desired velocity. The linear effort value requires a constant bias value in order to maintain a positive brake command when the remaining terms are zero. The throttle and brake commands are then calculated as

$$\begin{aligned} T_{effort} &= \begin{cases} k_t L_{effort} & \text{if } L_{effort} > 0 \\ 0 & \text{otherwise} \end{cases} \\ B_{effort} &= \begin{cases} k_b L_{effort} & \text{if } L_{effort} < 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} . \tag{4.16}$$

The constant control parameters used for the RD implementation on the NaviGator are:

$$k_p = 15.0,\ k_i = 6.7,\ k_d = 20.0,\ k_{ff} = 15.0,\ b_{effort} = -57.5,\ k_t = 0.65,\ k_b = -1.3,$$

and to prevent integral windup, the velocity error integrator is limited to ± 4.9. This keeps the vehicle from large over or undershooting of the desired velocity, if it is greatly disturbed, from terrain or other unexpected external forces.

After the PID controller is finished determining the steering and throttle commands, the complete wrench effort structure, containing the steering, throttle and brake commands, is ready to be sent on to the PD. This step is done by wrapping the values into the JAUS set desired wrench effort message, and then transmitting the corresponding data packet to the PD. This functionality is provided through the data structures and API functions linked to the RD application by the CIMAR JAUS libraries discussed above.

Faults and Failure Modes

The Reactive Driver control loop has several built-in fault detections, for conditions that may persist and can cause the system to fail. The first condition occurs when an obstacle or group of obstacles causes the planning search not to be able to find a solution trajectory. In this case the vehicle would inevitably have to collide with the obstruction if it were to continue on course. In order to avoid this possibility, the control loop detects the obstruction by observing that the HRHC solution passes through an obstacle, and slows the vehicle down. By doing this, the vehicle is free to make more evasive maneuvers because it is has more mobility at slower speed. In the case where the vehicle continues to slow until it comes to a complete stop, and still cannot find an unobstructed solution, then the RD goes into a fault state called “Blocked”. This means that the vehicle is essentially stuck in place and must either wait for the obstruction to clear out of the way or it must reverse to find a different course.

The Blocked condition tends to occur more than other errors in practice and testing, especially during navigation in harsh environments, such as off-road terrain. The causes for these blockages usually are false positive obstacles detected by one or more of the system’s Smart

Sensors, while operating on a narrow road or corridor. This forces the system to believe it cannot find an unobstructed solution, while in reality, there is nothing in the way.To overcome this problem in practice, a “Nudge” mode was built-in to the control loop, where the system would move forward slightly after being Blocked for a short period of time. Most of the time this had the effect of clearing the false positive in the traversability grid, and the NaviGator would continue on course. Often a light obstacle, such as a high patch of grass, or stick would cause the blockage and the vehicle would simply push over it unharmed.

Another failure mode may occur when there is some coupling between the vehicle dynamics and the traversability grid, input path, or goal point. This problem was discovered during an experimental implementation of the RD component on an Air Force Research Laboratory (AFRL) autonomous vehicle. In that experiment, the goal point for the HRHC search was set to be a fixed distance from the vehicle, and explicitly dependant upon its instantaneous heading. This distance and heading dependency caused the planner to repeatedly obtain the same trajectory and control solution. In this case, the initial control action was often opposite the desired motion. The first control command in the solution sequence would usually be intended by the planner to make a slight course correction, in order to find a more desirable path. This made the HRHC solution unpredictable and caused the vehicle to be unstable. The problem was corrected by removing the explicit dependence on the vehicles heading when calculating the goal point, thus allowing the algorithm to obtain a unique solution upon the following control loop.

Lastly, this implementation of a Receding Horizon Controller does not make use of a locally stabilizing control law. They are often used in RHC to drive the system inside of the goal region, and also in the stability analysis, to determine a terminal state cost penalty. The reason for this is that the system is not designed to ever reach the goal region. Upon each control loop

the goal region is recalculated at a point further along the desired trajectory than in the previous iteration. This cycle continues until the vehicle reaches the end of the path. In this scenario, the size of the goal region tends to be much smaller than the scale of the entire motion path, so it is reasonable to simply stop the vehicle at any point within the goal region, rather than driving it as close as possible to the goal point. Therefore, in practice the vehicle is simply halted, and the steering command is set to zero, when the goal region is reached, and so no locally executed control law is needed.

Conclusions

This Chapter has detailed the application and implementation of the theoretical concepts introduced in Chapter 3, and also introduced some new thoughts and considerations for obstacle avoidance in the optimization problem. Specifically addressed in the implementation overview, is the Reactive Driver component of the NaviGator's control system. This component employs the new and novel Heuristic Receding Horizon Controller, introduced in this dissertation as a means to simultaneously plan and control an autonomous vehicle through a cluttered environment.

As highlighted in the fourth section, there are a number of steps and procedures that must be addressed before execution of the HRHC algorithm. Some of these steps require ad hoc implementations, such as determining the goal region for the search. Also, there are some steps which require optimized data structures, and advanced knowledge of computing methods, such as the heap stack design, used here for the A* implementation.

The new planning and control method, which has been shown to unify two tasks into one, comes with some disadvantages as well. The fifth section, has mentioned some of the known faults and failure possibilities that this technique may present. Although, many are avoidable and

predictable, there is an inherent unknown when working with such a new technology that still requires a sufficient amount of testing and learning, in order to truly apply it in practice.

The following chapter is dedicated to some of the testing that has been done on this new and novel technique. It highlights and discusses the data and results collected, which help to support this thesis, and demonstrate the capabilities of both the theory and implementation discussed.

Table 4-1: RD's ready state control loop step by step procedure.

Step #	Step Procedure
1	Update information and local variables from incoming service connection data streams.
2	Check for system communications or PD faults. If any exist, then switch the RD into the Emergency State.
3	Check operator Run / Pause command. If Pause, then switch the RD into the Standby State
4	Determine the current path segment which the vehicle is attempting to navigate, based upon the systems newly updated position value.
5	Determine the current instantaneous desired speed from the RD's input motion path file.
6	Calculate the current goal region for the Receding Horizon Controller, based upon the input motion path, the planning time horizon, and the current desired speed.
7	Dilate the traversability grid received from the Smart Arbiter into a new grid which will be used for planning by the RHC routine.
8	Execute the HRHC algorithm given the current desired speed, goal region, vehicle state, and dilated traversability grid. Obtain the desired steering command based on this algorithm.
9	Execute speed PID control loop in order to calculate the current desired throttle and brake commands.
10	Wrap the current desired steering, throttle and brake commands into a JAUS set wrench effort message, and send the message to the Primitive Driver for execution.
11	Repeat procedure, starting at Step 1.

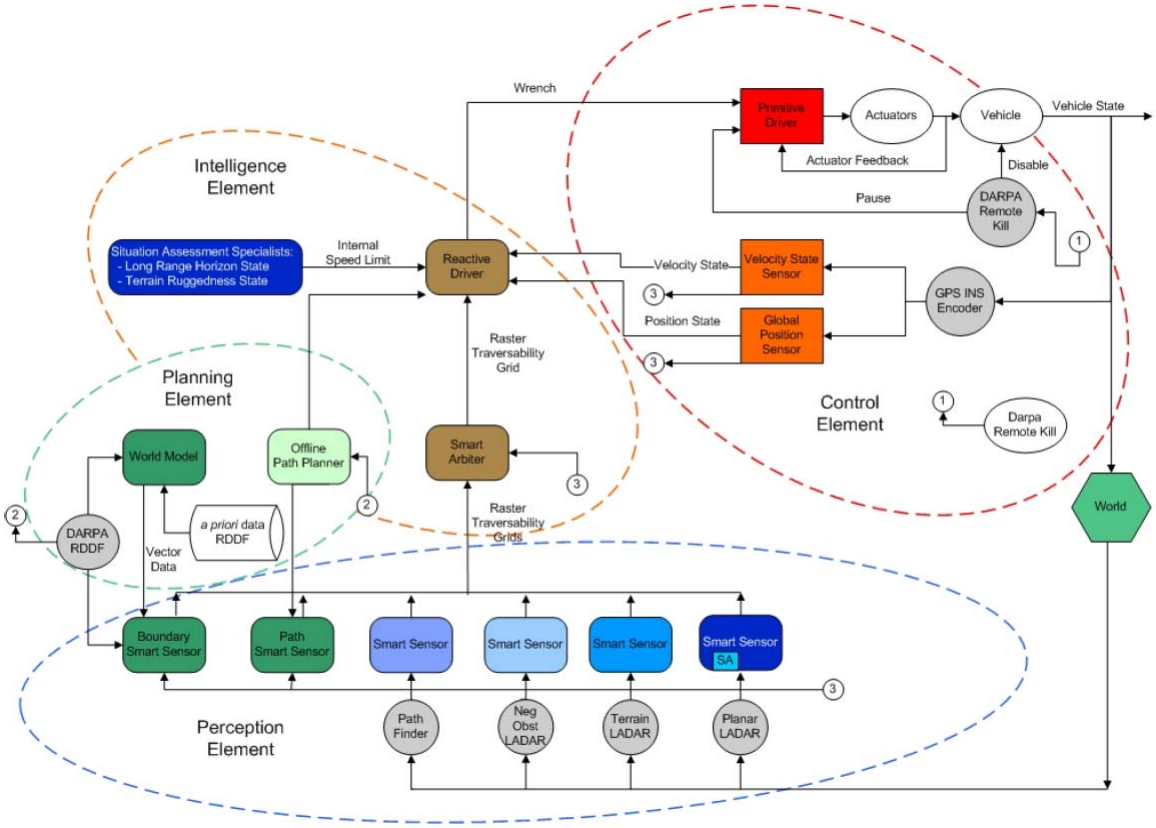

Figure 4-1: The NaviGator high level control system block diagram. The system is made up of four key elements, highlighted in red, orange, green and blue. They are the: control, intelligence, planning, and perception elements respectively.

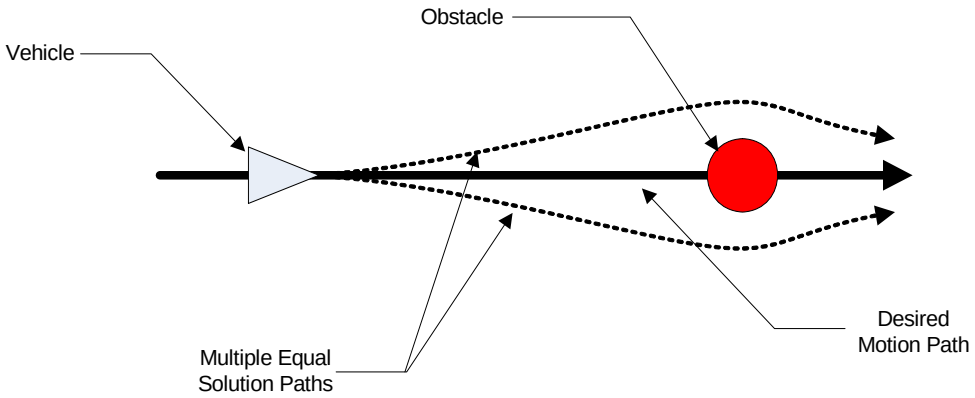

Figure 4-2: Simple obstacle avoidance case, showing a vehicle attempting to track a straight line and a single obstacle existing on the center of that line. This case intuitively shows that there may exist multiple and equivalent minimum cost state trajectories, yielding different but equal control input sequences as a result of the RHC optimization process.

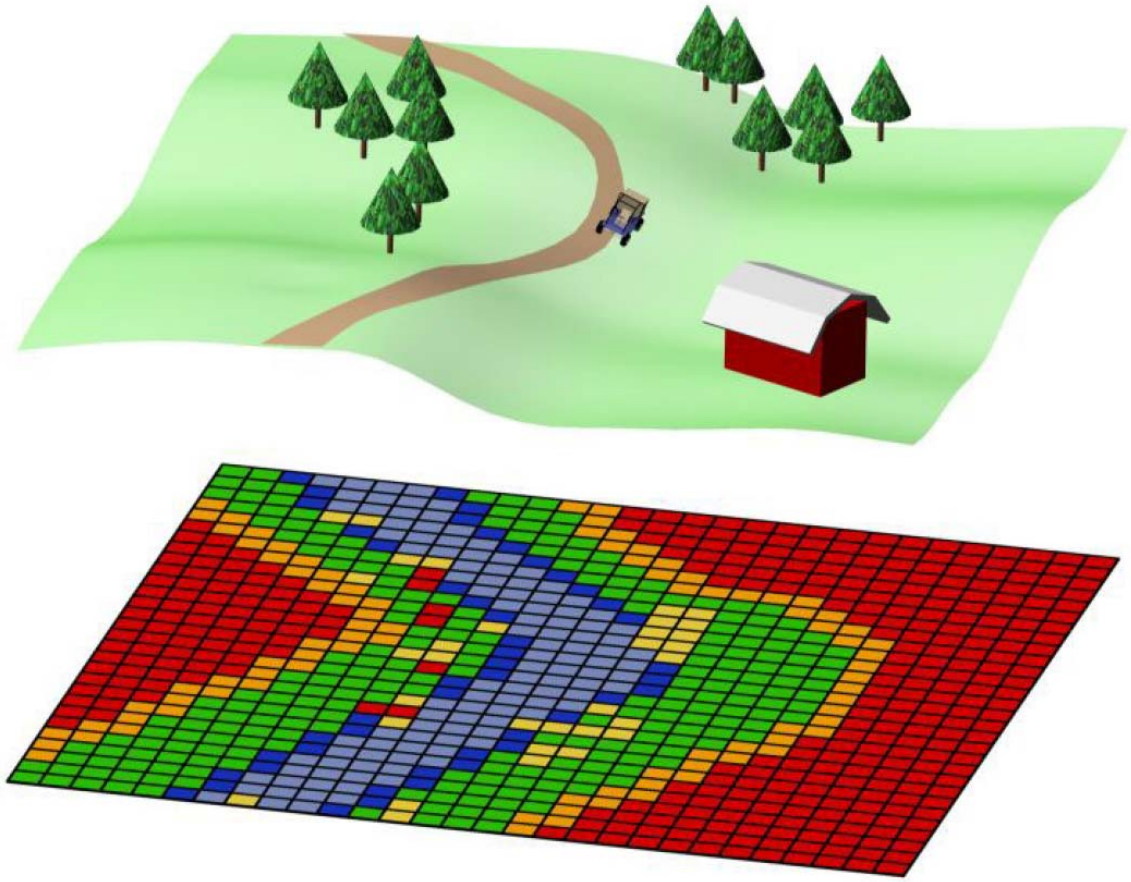

Figure 4-3: The traversability grid concept. The local environment, depicted (above), with a corresponding traversability grid (below). Both images are conceptual representations of both the real world local environment around the vehicle at the center, and its mapped grid values respectively.

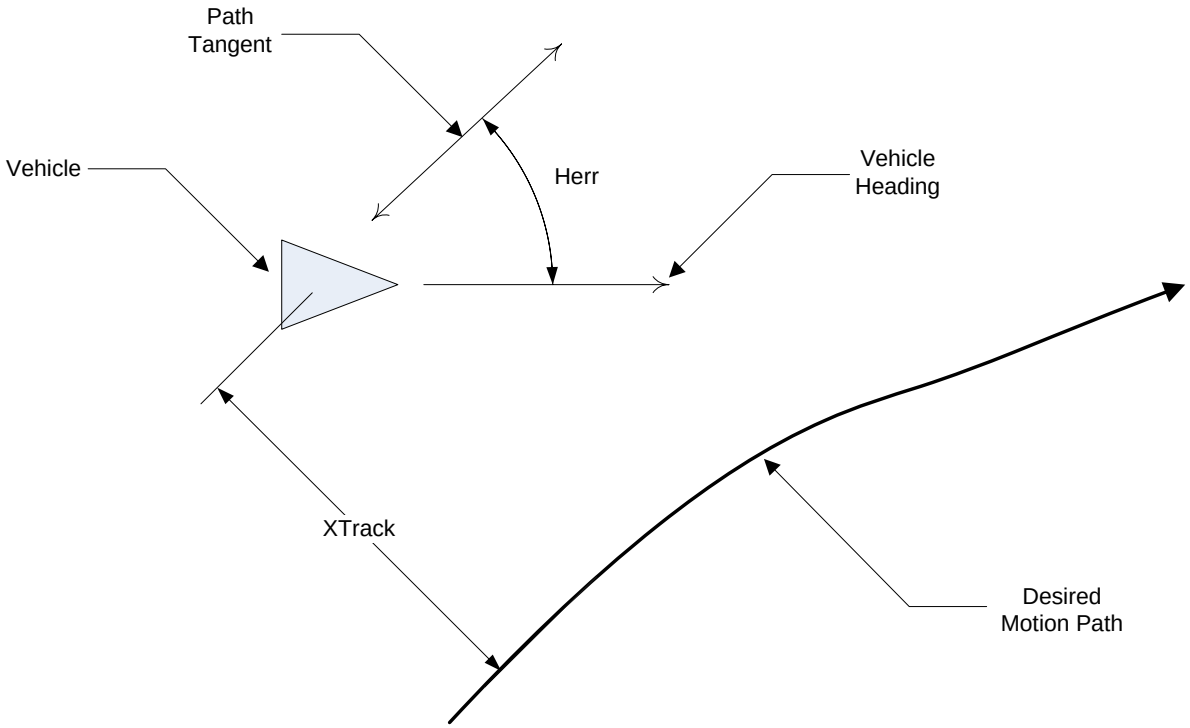

Figure 4-4: Path tracking error system. The tracking metric is made up of two measurements. The cross track error measurement, XTrack, is the perpendicular position error of the vehicle with respect to the path. The heading error measurement, Herr, is the heading error angle of the vehicle with respect to a path tangent.

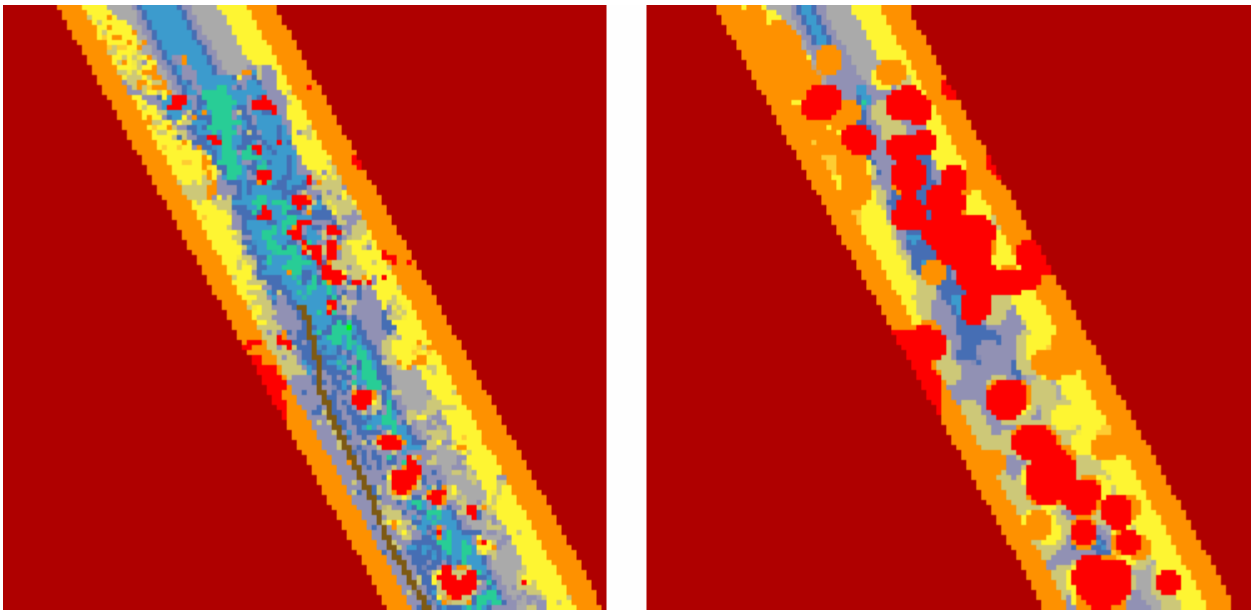

Figure 4-5: Traversability grid dilation before (left) and after (right). The grid on the right is dilated with a circular kernel with radius 2.5 pixels.

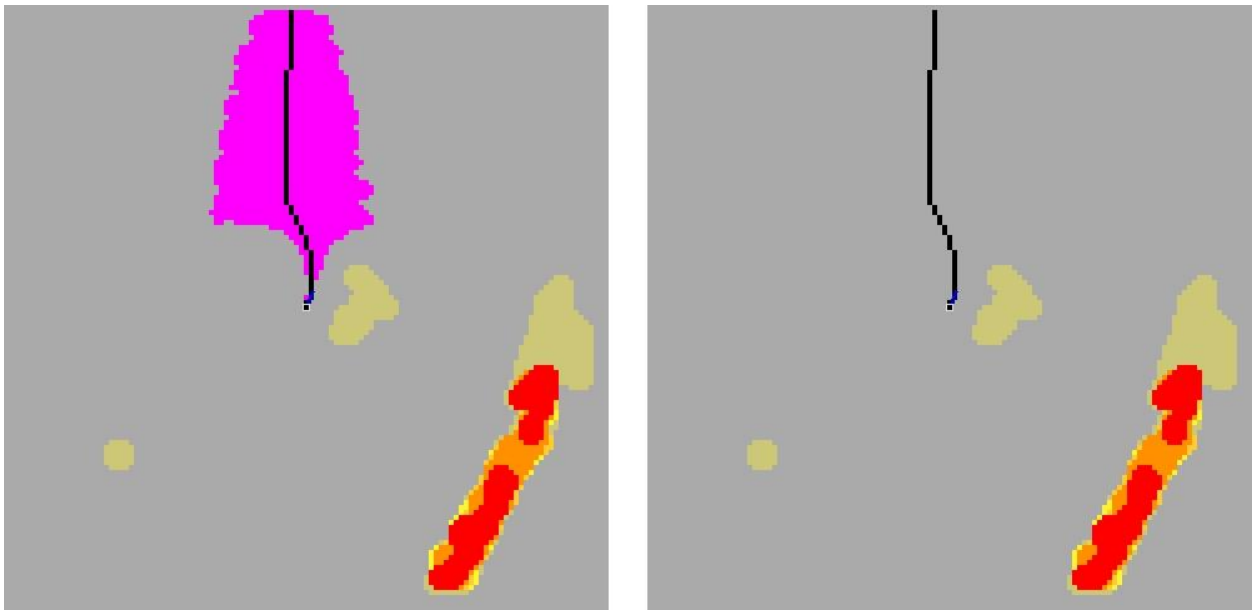

Figure 4-6: Planning and control search area, sample traversability grid outputs. The pink region (left) shows the area searched by the A* optimization routine, in order to determine the solution sequence trajectory shown in black. The image on the right shows the optimal predicted trajectory with no obstacles surrounding the path sequence.

CHAPTER 5
TESTING AND RESULTS

The receding horizon controller described in this dissertation has been implemented and tested in several different forms. The first version implemented was designed for the 2005 DARPA Grand Challenge competition. Several lessons learned were gained from working with and testing that implementation. The most important one them is that the motion stability of the RHC is highly coupled to the system model and the optimality of the calculated control sequence. This unfortunately means that there are few gains or parameters that can be tuned to adjust control stability. Simply put, if the system model is drastically incorrect or the optimization routine cannot find an admissible solution then the system will be inherently unstable. At that point, either a new more accurate model must be introduced and or the optimization algorithm must be enhanced.

The second version was implemented on an Air Force Research Laboratory (AFRL) vehicle at Tyndall AFB. The vehicle, known as the Land-Tamer, is a skid-steered hydraulically actuated platform. Therefore, the vehicle model used for the RHC implementation was significantly different from the one used for the NaviGator vehicle. Several lessons were learned from this implementation. The main one discovered was the state-goal coupling instability failure, which was discussed in Section 5 of the previous chapter.

The latest version follows directly from the discussion of the previous chapter and is the one being tested and reported on here. This implementation is the most inline with the theoretical approach detailed in Chapter 3. The CIMAR NaviGator vehicle and the computing machines within will serve as the general test bed and provide a single platform to eliminate any variation between implementations on different machines. It is assumed that if the claims made within this dissertation can be supported by implementation and testing on the NaviGator vehicle, then they

would in turn be validated for other similar AGVs. Although, it should be understood that any methods tested here would almost certainly need to be modified and or tuned to support the vehicles upon which they are operating.

Before infield testing of the implemented RHC algorithm, a test plan was established for organizational purposes. The first section of this Chapter outlines and explains the test plan for data collection and analysis of results. The plan has since been conducted and a post test review is given in the second section. Finally, the data results are presented and some conclusions are drawn in the third section.

Test Plan

The following test plan establishes and outlines a means to collect experimental data and results that can either support or refute the primary new and novel concepts of this dissertation. The critical concept follows from Remark 1 of Chapter 3, which claims that RHC applied to an AGV inherently unifies the planning and control tasks. Traditionally these tasks have been seen as separate problems for autonomous vehicles.

In order to support this claim, information and evidence must be collected that shows a single RHC process accomplishing classic controller-like capabilities such as: stabilization and regulation onto a predetermined motion structure, while also achieving convergence characteristics that can be identified by typical performance metrics. This means that measurements defining the system's time response to step, ramp, and other motion commands, must be obtainable. In addition, separate tests must be conducted that show the same exact RHC task demonstrating planner-like capabilities. These include abilities such as: obstacle avoidance, and favoring desirable terrain. Data that corroborate these behaviors, such as object clearance distance and trajectory departure times, should also be presented.

One item of critical importance in testing these claims is the use of baselines or controls against which the aforementioned results can be compared. One valid baseline would be use of a classic AGV motion controller. Pure Pursuit [COU92], Vector Pursuit [WIT00], or a simple PID controller, have all been implemented in the past with success. A comparison of the RHC algorithm's performance against one of these algorithms implemented on the same vehicle would offer a useful contrast to relatively score the new control algorithm's value. If in addition, the RHC algorithm can be shown to achieve planner-like behaviors then clearly some value has been added to the overall control; an ability which by nature could not be provided by any traditional motion control scheme. To facilitate organization of the different tests that will be conducted Table 5-1 details the design, setup, procedures, and data collection for each part.

Test Review

The tests detailed in the previous section, were conducted on Friday September 15th 2006, at the Gainesville Raceway Road Course (Figure 5-1). This location was selected because it offers a controlled on-road operating environment for the vehicle, and its seclusion makes it a very safe place to run the NaviGator.

Since the NaviGator is a large and heavy vehicle, safety is very important during its operation. If the vehicle becomes unstable, or goes out of control, it is critical that no person is nearby, because the behavior of the system would be unknown. The road course allows a robust test to be conducted, with few observers and operators needing to be present. Also, in the event that a problem does occur, the NaviGator is equipped with a rugged wireless kill switch system, which shuts down the engine, releases its emergency brake, thus assuring that it will come to a stop. The kill switch system also has a run / pause signal which is used to start, stop, and pause mission execution from a safe distance. At the Gainesville Raceway, the vehicle's kill switch can

be controlled from a single static base station. This is because there is continuous line-of-sight to the NaviGator no matter where it is on the course.

The test location also offers a near limitless variation of path geometry that may be constructed for any given test run. There are a number of different smooth curved track segments, many with different radii, and also long straight-aways, both of which allow for good testing of tracking performance, and meet the criteria of the test design above.

More importantly, the track allows the observers and operators to visually gauge the vehicle's performance in real-time. Since the designed path geometry is known ahead of time, it is clear that by observing the vehicle on-road, it is executing the correct mission. A motion path executed over an area with no clear landmarks or road geometry, makes mission observation uncertain at runtime, and only by observing tracking data, can it be made clear whether or not the vehicle executed the mission correctly.

The conducted tests required several days of setup and preparation prior to being able to run them and collect valid data. The first step in this setup process was to create a map of the racetrack. This was done by logging global position system (GPS) data of the NaviGator while it was driven manually around all of the different parts of the course. Once collected, the data was post processed and loaded into Mobius, an Operator Control Unit (OCU) software application developed by Autonomous Systems Inc. Within Mobius, the GPS data of the track was used to create a map, so that motion path segments of the test mission could be drawn by hand over the computer modeled track. The path segments drawn were designed to follow the test plan intent and layout detailed in

Table 5-1. These segments were then saved to a flat file, in the format accepted by the NaviGator's RD component, and uploaded to the vehicle control computing node.

The conditions during test runs were nominal. The weather was fair, allowing for the mechanical systems of the vehicle to run without any problems, such as overheating or rain interference with system sensors. However, one problem presented with the onboard GPS system throughout the day. This was observed when the vehicle was noticed to drive along a different lateral position on the road course during each individual test run, even though the mission path file was held constant. This behavior is typical because the GPS solution position will tend to drift slowly throughout the day as the GPS satellite constellation continuous to change overhead. However, the GPS solution on the day of the test drifted to an extent which caused the vehicle to drive adjacent to, and not on the road. Although, it remained clear that the vehicle was still attempting to drive, the correct path as sensed by GPS. The problem was analyzed and isolated to the onboard position system, because the control error never reached a value greater than a few meters, while at the same time the vehicle was visually observed to be off track by approximately 10 meters. Therefore, the problem had to be within the GPS data.

The obstacle avoidance portion of the tests was also successful. However, since the vehicle's position estimate continued to change over time, it was difficult to place the obstructions along the path so that the vehicle would encounter then head-on during any given run. One lesson learned from this, is that it is best to use very large obstructions when testing OA with position uncertainty. The obstacles used were construction barrels approximately 0.6 meters in diameter, which was found to be somewhat too small to test with while position system errors are on the order of several meters. Nevertheless, the vehicle was found to avoid the obstacles, when encountered, and the data presented in the following section supports this find.

Test Results

The first step in the test given the plan above is to design a path circuit for motion tracking. As aforementioned, this was done using the Mobius software tool. The path circuit designed for

the tests indeed does satisfy the requirements presented in the test plan. It is made up of segments both straight, and curved, with the curved portions having varying radii. The road course at the test site greatly facilitated this design. The segments were easily mapped over a subset of the track and lie near the center of the actual road pavement.

The decided upon path segments and their geometry is depicted in Figure 5-2. As shown there are 11 segments, 5 straight and 6 curved; they are numbered in order of intended motion tracking. The curved segments tend to decrease in radius as the circuit progresses from start to finish, and thus making tracking more and more difficult as the vehicle pursues the mission. The last segment is a large arc included simply to bring the vehicle back to its starting position on the course. Also, notice that there is a discontinuity between segments number six and seven. This was purposely built into the circuit in order to test the system's response to an instantaneous step in cross track error. Since both segments have the same orientation, the cross track error is isolated and its response can be measured.

The data making up the path file for the test is summarized in Table 5-2. For each of the 11 segments, the starting point latitude and longitude are given, along with the endpoint latitude and longitude. The radius values shown in the table were calculated as the multiplicative inverse of the segment curvature specified in the path definition file. Also each segment has an associated desired speed value, which is not shown in the table, but was set to 4.5 mps for the test. This is a nominal speed for the NaviGator and allowed for the effect of speed variations in testing to be minimized.

The first series of tests are intended to establish tracking results for the designed path circuit using a classic vehicle motion controller on the NaviGator. This is done in order to

determine a set of baseline metrics that may be compared to the measurements found for the RHC algorithm that is executed in the RD component.

The classic method decided upon for this test is a simple PD controller that regulates the two tracking states: cross track error, and heading error. The controller is used to determine the vehicle steering command, while it is operating at a constant speed along the path. By regulating the two tracking signals to zero, the vehicle is guaranteed to track the desired path.

The controller was implemented within a component on the NaviGator, and took the place of the RD within the vehicle's overall control system. The component is called the Global Path Segment Driver (GPSD), and takes as its input, the same global path segment file that is used for RD execution. Like the RD, it uses feedback from the GPOS and VSS components to measure the vehicle's state in order to generate a control wrench message that is commanded to the PD component. Unlike the RD, this component does not receive wrench feedback from the PD, nor does it receive any traversability grids from the vehicle's Smart Arbiter. For consistency, the PID speed controller within the GPSD was kept to be an identical copy of the one used within the RD component, thus canceling any effect speed control might have on the performance of the two controllers.

The PD control algorithm is essentially two controllers summed together, which determine a wrench value somewhere between -100 and 100% steering effort. It is also modified slightly with a feed forward path curvature term in order to assist the controller with tracking curved path segments. The steering effort control value S_{effort} is thus calculated by:

$$S_{effort} = k_{xtp} XTerr + k_{xtd} \frac{d\,XTerr}{dt} + k_{hp} Herr + k_{hd} \frac{d\,Herr}{dt} + k_{ff} s_k . \qquad (5.1)$$

Where $k_{xtp}, k_{xtd}, k_{hp}, k_{hd}$ are the PD gains for the cross track and heading error respectively, k_{ff} is the curvature feed forward gain, and s_k is the current path segment curvature. The gain values tuned into the NaviGator for this controller are:

$$k_{xtp} = -7.0,\ k_{xtd} = -2.0,\ k_{hp} = -45.0,\ k_{hd} = -10.0,\ k_{ff} = 625.0,$$

and were determined qualitatively to produce stable desirable tracking motion.

Three test runs were conducted, measured and logged for this controller. The data collected was logged at a rate of 10 Hz, the same rate at which the controller is executed. The first run was intended to measure the controller's tracking performance along the path, given no initial significant error in either cross track or heading. It was setup by positioning the vehicle near the start point of the first path segment, and then switching the automatic control system on. The radio kill switch system mentioned above was used to start the NaviGator from a safe distance.

Figure 5-3 shows the tracking position data logged from the first run of the GPSD. Due to the scale of the image, it is difficult to observe the fine details of the tracking performance of the vehicle. However, there are several distinguishable features that can offer some insight. In the plot, it can be seen that the vehicle begins slightly off track and then quickly stabilizes and converges onto the path. During most of the initial tracking, the vehicle is too close to the path to notice any difference between the two data sets, but after the 6th segment it is clear that the planned instantaneous step in cross track error is quickly regulated back close to zero. Lastly, notice that the vehicle tracking performance degrades significantly around the two tight turn segments 8 and 9. This is because the turn radii are very close to the NaviGator's minimum turning radius, and it becomes very difficult for the controller to regulate the vehicle while it is so close to its operating limit. Nonlinear dynamic effects, such as input saturation tend to increase in their impact on performance during these scenarios. Also, the inherent delay in

steering angle, due to its natural rate limit, keeps the steering wheels from getting to the correct tracking position as soon as the segments are encountered. This is a classic problem with front wheel steered vehicles and their dynamical constraints. If the path being tracked is not second order continuous, then the vehicle is required to come to a stop at the discontinuity, so that it may turn its wheels to the correct position. Since the path segments designed here are continuous in both position, and heading, but not curvature, this phenomenon presents itself in the test.

An anomaly discovered during data analysis and post processing can also be observed within the planned path itself at segments 8 and 9. Clearly there is a discontinuity between segments, although they were designed to be smooth. This is simply due to post processing, because the path data points were calculated from the vehicle's location, cross track error, and heading. In the controller software the path segment transition happened prior to the intersection of the two segments and so a discontinuity is observed because the segments do not share the same curvature or center point location.

Since the scale of the complete path circuit is too large to observe the full detail of tracking performance, a better set of data to visualize are the heading and cross track error signals. A plot of the heading error for the first test run of the GPSD is provided in Figure 5-4, it clearly shows more detail of the regulation performance for the PD controller. At this scale it is possible to observe the quantized nature of the heading feedback signal, due to resolution of the system's heading sensor, the discrete values of the signal are identifiable within the plot. Clearly since the two tracking signals are coupled dynamically, the plot of the system's cross track error appears similar, and is given by Figure 5-5.

Notice that both signals indicate a large initial tracking error, a large disturbance at approximately 100 sec (due to the discontinuity between segments 6 and 7), and another large

disturbance at 125 sec (due to the tight turn segments 8 and 9). Also, both signals settle out and remain close to zero for the last sixty seconds. Rather than display and analyze both heading and cross track error for each test run, their inherent coupling allows only one to be studied to support the thesis. Therefore, for brevity, analysis of all test runs is focused only on cross track error, for this dissertation. Its resolution on the scale analyzed here is finer than the heading error's, so it allows for clearer distinctions to be made.

The second run of the GPSD tested its response to an intended very large initial cross track error. The purpose of this was to observe the PD controller's reaction to an error value closer to the point of its operating envelope. Since the PD is a linear control system operating on a nonlinear system, its control region only a local neighborhood of the error system origin. Outside of that region the control system will inherently be unstable, and the system will fail. It is incapable of globally stabilizing the vehicle onto the path, or in other words, it cannot bring the vehicle to the path from any arbitrary point. A simple case can justify this concept. As the cross track error increases from zero to infinity, at some point it will completely dominate the remaining terms in the control equation (5.1). At this point, the steering effort will be saturated in one direction, and the vehicle will simply drive in a never ending circle, because the cross track error will never drop below the saturation value.

The cross track error signal for the second run is depicted in Figure 5-6. Here the large initial cross track is shown to be -25 meters. Regardless of this, the system is indeed within its operating limits because the tracking error is quickly regulated and stabilized back close to zero. However, due to the controller's linear capability, there is a large overshot of the path (7 meters, or about 30%) and again an undershot (2.5 meters) before it is reacquired. From that point on in the path mission, the tracking performance is very similar to the first test run. The disturbance

between segments 6 and 7 is rejected, and the error peaks to approximately 3 meters on the sharp turn segment 8 and 9. The large overshoot in this test run is highly undesirable since the vehicle traveled far off of the intended path after it was already reached.

A final test run of the GPSD was conducted with an even larger initial cross track error than the second run. This was executed to demonstrate the unstable nature of the control system while operating outside of its acceptable range, and to observe its behavior during this mode. The expected behavior was that the system would drive in a continuous circle, however that was not observed. Instead, the vehicle turned almost completely around and started driving a direction opposite than what the path desired. This is most likely because the vehicle was still within a range that did not saturate the steering command, but was enough to create an unstable and unpredictable behavior. In Figure 5-7,notice that the cross track error begins to converge but is then interrupted and only reaches a minimum absolute value around 20 meters.

The output signals from each test run were also recorded for control system analysis. Figure 5-8 provides the three signals: steering, throttle, and brake. All are mapped within the same plot because they all have a similar scale. Steering exists from -100 to 100% effort, while throttle and brake have a range of 0 to 100% effort. The steering signal is negative for left-hand turns and positive for right-hand turns. The steering signal clearly indicates all of the maneuvers and features encountered during the run. There is an initial response to the off-track starting position, each turn segment can be clearly identified from the large steps in the command, which occur due to the curvature feed-forward term in the controller, and the cross track step at segments 6 and 7 is seen as a step response in the steering shortly before the 100 sec time mark. Interestingly, there appears to be a slight bias in the steering control for long straight segments. This is most likely due to a small misalignment in the steering actuator system.

Speed control for the GPSD system worked well during the tests. The velocity control system response for the first GPSD test run can be seen in Figure 5-9. As indicated there are many disturbances in the signal, even though the command is held constant, the speed does not appear to reach a steady state. This occurs for several reasons. First, the current path segment geometry is changing regularly, so the steering wheels must turn often and thus upset the external forces acting on the vehicle. Second, the throttle, engine, and drive train dynamics are highly nonlinear. Continuous gear switching in the automatic transmission is one nonlinear effect, for example, which can cause the system to change response. Also, the resolution of the speed measurement is only around 0.1 mps, this may prevent the control system from any fine tuning that is required to achieve steady state. Despite this behavior, the measured speed remains within 0.5 mps of the desired most of the time. This is a very acceptable bound in practice.

In summary of test part 1, the set of runs conducted with the GPSD component executing the PD controller described, yielded a complete set of baseline data needed for comparison to the RD algorithm. The controller performed as expected, stabilizing the vehicle's motion onto the *a priori* path structure, and performing better on wide turns than on sharp turns. The expected unstable behavior for a large cross track error was also observed in the third run, and offers some insight to the limits of the algorithm.

Test part 2 requires runs identical to the nominal cases of part 1 to be conducted using the RD component, which executes the new and novel receding horizon controller introduced in this dissertation. As a brief review, the RHC algorithm implemented on the NaviGator uses feedback information from the system state and environmental sensors, to simultaneously plan and control the vehicle's motion on track and around obstacles. The output of the RD is a wrench command, containing the three steering, throttle and brake effort values, and is identical to the output format

of the GPSD. The test runs conducted here in part 2 measure the algorithm's ability to regulate the path tracking of the segments given in Figure 5-2.

For part 2, two test runs were conducted, measured and logged for this controller. The data collected was recorded at a rate of 10 Hz, the same rate at which the RD receding horizon controller is executed. Also like before, the first run was intended to measure the controller's tracking performance along the path, given no significant cross track or heading error.

Figure 5-10 shows the tracking position data logged from the first run of the RD. As before, the scale of the image makes it difficult to observe the fine details of tracking performance. However, the same distinguishable features are present after the 6^{th} segment where it is clear that the planned step in cross track error is quickly regulated back close to zero. Also, the vehicle tracking performance degrades significantly around the two tight turn segments 8 and 9, as expected and similar to the linear controller. The main difference between the GPSD and the RD observed here is that the RD tends to have a more prominent steady state cross track error. This is due to the nonlinear nature of the controller, input quantization, and system model inaccuracies. However, the performance overall remains quite acceptable for path tracking and regulation.

The cross track error data for this run is a much better indicator of controller performance at the vehicle scale. Shown in Figure 5-11, the error signal interestingly corresponds closely to what was observed in the PD controller signal, with the exception that steady state error is larger. However certain performance features are almost identical to those observed in Figure 5-5, and Figure 5-6. For example, the initial error quickly converges down to a value within 1 meter, the 6 to 7 segment transition step and its response is clearly observable around 100 seconds, and like the PD controller, the RHC has difficulty maintaining close tracking during the sharp turns. The

peak error value around 3 m during segments 8 and 9 is analogous to the peak error observed in the linear controller. This data shows concretely, especially when compared to Figure 5-5, that there is no major tracking performance difference between the two controllers. The only distinguishable difference remains the cross track steady state error, which although is larger for the RD, is within acceptable limits for the task. Most likely it could be reduced further with some additional performance tuning.

A more noticeable distinction between the PD and RHC controllers is identifiable in the second RD run data (see Figure 5-12). In this instance, the RD was given a large initial cross track error, approximately equal to that of the second run of the GPSD, about 25 meters. In this case, there is still an overshoot of the path, but not nearly as much as the one encountered for the GPSD. The overshoot seen here is approximately 1.5 meters, which is still within acceptable operating limits. This ability makes the RHC controller much more desirable than the PD controller, since fast and stable path reacquisition is one of the most critical performance qualities of an autonomous vehicle control system. After this portion of the test run, the performance remains nominal and similar to that seen in each of the previous stable runs.

Figure 5-13 presents the output performance for the RHC controller. Here some significant differences in control can be observed between the classic PD controller and the RHC algorithm. The first noticeable difference is that the RHC steering output tends to oscillate in a square wave like fashion, especially during the wide turn segment times. This is caused by the artificial input quantization that must be done in order to allow the HRHC algorithm to find a solution in real-time. The quantization does not produce a fixed set of discrete states for the steering value to occupy, because it is based off of the instantaneous steering feedback from the primitive driver component. Thus the quantization is recomputed *ad hoc*, within the node expansion function of

the A* search. Another clear distinction is that the RD uses much more command authority (steering range) than the PD controller, specifically during the initial convergence, and the sharp turning period. This is largely attributable to the lack of steering cost penalization within the value function of the RHC. However, this effect is not undesirable in practice since it allows for faster tracking and regulation of path motion. Conversely, the RD uses less control authority than the GPSD for small disturbance instances, such as the cross track step at segments 6 and 7. This is due to its higher steady state error tolerance, and lack of convergence effort within its goal region.

For equality the speed control system performance of the vehicle during RD execution is presented in Figure 5-14. The plot is given for the first and nominal run of the RD, as with the GPSD plot. Similar to the previous speed controller performance graph, this figure shows the speed remaining with 0.5 mps of the desired speed, most of the time. Interestingly, the speed tends to converge during the later portion of the run. This is an example of the unpredictable nature of a nonlinear control system within its bounded stability region. It could have been caused by anything from a constant gear selection in the automatic transmission, to an activation of the vehicle's onboard air conditioner, which is coupled to the engine dynamics through power generation equipment. Since these causes are not measured or modeled, they cannot and are not explicitly compensated for in the controller.

As a final controlled comparison and summary of tests part 1 and 2, the response times, percent overshoot, settling times, and steady state errors, were recorded for the cross track error step between segments 6 and 7, for both the GPSD runs 1 and 2, and the RD runs 1 and 2. Table 5-3 contains the values determined for each measurement. This side by side comparison analysis of the four step responses strongly supports the hypothesis that the RHC algorithm can

effectively regulate an autonomous ground vehicle onto a predetermined motion structure. While there appear to be some quantitative tradeoffs between the two controllers, it is evident from both the plots above and the table, that new RHC controller response is stable, effective and its response is able to be measured in the same way as a classical controller. Clearly by tuning it is possible to change the numerical results presented here. Therefore their purpose is only to show that the two controllers are comparable and within some tolerance of one another, which it indeed does.

The third test part is intended to demonstrate the Obstacle Avoidance (OA) capability of the RHC algorithm, something which cannot be provided by a classic style controller, and to offer an analysis of the collected OA data. The data was collected over a single run of the RHC controller on the same test track as the previous two parts. Along with the signal data typically collected on each run, traversability grid images, which are output by the RD as a development and debugging tool, were recorded. These images were logged remotely by a standalone visualization component at a rate of 3 Hz, an arbitrary value.

Unfortunately, post processing of the tracking data from this test revealed significant position system problems during the run. There are several visible jumps in the vehicle's estimated position, and as a result, it is difficult to detect where in time the vehicle deviates from the prescribed path in order to avoid an obstruction. Nevertheless, there were a total of four obstacles placed on the path prior to the run, and the NaviGator was able to avoid them. The four objects were large construction barrels, which are easily detected and mapped into the traversability grid by the NaviGator's Smart Sensor components. The logged traversability grid data shows the avoidance much more apparently than the position and error signal data.

As in the previous two test parts, an overview of the vehicle's track position data is presented in Figure 5-15. Here it can be seen that the vehicle deviates from the path at the four obstacle locations, each of which is called out. Position system uncertainties are clearly present, especially around segment 7, which indicates a strong estimate drive to the North.

The detail of the run is identified in Figure 5-16. Again the four obstacle avoidance maneuvers are called out in the diagram. In these portions of the mission, the vehicle clearly deviates from the intended tracking, as anticipated. Also, the large jumps in position estimate are called out and circled in blue. These are not control system related, and instead are caused by solution discrepancies within one of the two onboard GPS receivers.

The obstacle avoidance of each of the four barrels is most apparent in the traversability grid images, as mentioned. Figure 5-17 provides these grid images, and indicates the results of the RHC optimization routine; at or near the time each object was passed by the NaviGator. The barrels appear as red or orange circles in each image, and were dilated by a radius of 3.5 pixels, so the vehicle would have enough clearance to maneuver around them during the run. Other large red or orange areas in the images indicate surrounding terrain or trees, and are not in the way of the vehicle while it is on track.

Some of the images have spotting artifacts in them which are attributable to instantaneous jumps in the position solution, which causes sensed data to be placed in the wrong grid location. However, since the position between the obstacles and vehicles is relative, their relative position in the grid remains constant, and the RHC algorithm is able to consistently steer the vehicle around the true obstacles.

Also, as discussed in the previous chapter, the pink region in the grid indicates the possible solution trajectories that were explored during optimization, and the black or white line shows

the final trajectory selected. The pink lines on the grid edges are new spaces which the vehicle is moving towards, but has yet to populate with any environment estimate.

In summary of test part 3, the NaviGator was given a path mission identical to the previous two tests. The vehicle performed as expected, avoiding all four of the encountered obstacles and also maintaining adequate path tracking while in free space. These data and results of this part therefore support the hypothesis claim that RHC applied to an autonomous ground vehicle allows planning and control to be done simultaneously.

The next chapter focuses on the overall results and conclusions which can be drawn from this dissertation. It also details some of the advanced concepts, which are outside of the scope here and also some future work that can be done to progress this research area.

Table 5-1: The receding horizon control autonomous vehicles test plan. The test plan is made up of a hypothesis and three main parts: one for control, another for the RHC algorithm itself, and lastly a part to test Obstacle Avoidance.

Part Label	Part Explanation
Test Purpose	Establish RHC algorithm's ability to unify planning and control for autonomous vehicle motion, and to compare the algorithm's performance against a classical linear controller.
Hypothesis	RHC applied to an autonomous vehicle inherently unifies the planning and control tasks, so that they may be executed simultaneously. This allows a single task to accomplish motion state regulation and also provide obstacle avoidance capability.
Expected Results	The RHC algorithm will exhibit robust tracking performance and will be less susceptible to large disturbances than the classic linear controller. However, the linear controller will probably prove to track better than RHC, under nominal conditions, because the RHC algorithm is inherently suboptimal due to input quantization and other factors such as system dynamics model inaccuracies. As an addition, the RHC method will also be able to avoid obstacles, whereas this behavior is impossible for the linear controller.
Part 1, Purpose	Measure classical linear path tracking controller performance as a control for the hypothesis.
Part 1, Design	The vehicle will attempt to track an obstacle free path using a classical motion controller. The path will be provided as a preset circuit made up of a variety of straight line segments and curved segments of different radii. The use of a circuit track will maximize the collection of data per given test run, which is beneficial since the time and preparation required to setup a single run is nontrivial. The circuit will be designed such that the system should be able to reach a stable steady state on one segment before the next is reached. For each new segment encountered on the path, there will be a discontinuity in curvature, heading, or cross track error. These discontinuities will allow for time domain step response measurements to be made. The test will be executed and logged at a nominal speed, in order to isolate the effect of velocity on tracking performance. The test will then be repeated with a large initial tracking error, in order to measure the controller's ability to reacquire the path.
Part 1, Logged Measurements	Error Signals: Cross Track Error, Heading Error, and Speed Error. Input Signals: Desired Vehicle Position, Heading, and Speed (all from input path). Output Signals: Steering, Throttle and Brake Actuator Commands.
Part 2, Purpose	Measure the RHC algorithm's path tracking control system performance.

Table 5-1: Continued.

Part Label	Part Explanation
Part 2, Design	The vehicle will attempt to track an obstacle free path using the RHC algorithm detailed in Chapter IV. All test tracks, speeds and procedures will be identical to that of part one in order to maintain equality between the two tests. The test will then be repeated with a large initial tracking error, in order to measure the RHC ability to reacquire the path.
Part 2, Logged Measurements	Error Signals: Cross Track / Transverse Error, Heading Error, and Speed Error. Input Signals: Desired Vehicle Position, Heading, and Speed (all from input path). Output Signals: Steering, Throttle and Brake Actuator Commands.
Part 3, Purpose	Measure the RHC algorithm's planning ability.
Test Part 3 Design	The vehicle will attempt to track the same path circuit as the previous to test parts with the addition of obstacles. The obstructions will be placed such that tracking the path on center would cause an impact, therefore the vehicle must depart from the current path segment in order to avoid collision with the obstacle. The obstacles used for this test will be lightweight construction barrels, which are easily detected by system sensors and safe in the event of a collision.
Test Part 3 Data	Error Signals: Cross Track / Transverse Error, Heading Error, and Speed Error. Input Signals: Desired Vehicle Position, Heading, and Speed (all from input path). Output Signals: Steering, Throttle and Brake Actuator Commands. Other data: Obstacle Clearance Distance (Minimum distance observed between vehicle and obstacles)

Table 5-2: Test path circuit specification data. The 11 path segments for the control system tests described in this Section are given with a start point, and end point latitude and longitude. The segment radius is calculated as the multiplicative inverse of the specified curvature. Positive curvature values indicate a leftward turning segment and negative values indicate a rightward turning segment.

Segment	Start Lat (deg)	Start Lon (deg)	End Lat (deg)	End Lon (deg)	Radius (m)
1	29.75262026	-82.26275871	29.75340236	-82.26275587	∞
2	29.75340236	-82.26275587	29.75376698	-82.26318436	40.0
3	29.75376698	-82.26318436	29.75336974	-82.26361331	41.7
4	29.75336760	-82.26361326	29.75268459	-82.26361854	∞
5	29.75268459	-82.26361854	29.75247728	-82.26383552	-23.3
6	29.75247705	-82.26384131	29.75248487	-82.26467643	∞
7	29.75250188	-82.26473602	29.75250472	-82.26565011	∞
8	29.75250472	-82.26565011	29.75240730	-82.26576274	10.9
9	29.75240730	-82.26576274	29.75231923	-82.26567174	9.8
10	29.75231897	-82.26566695	29.75228804	-82.26321990	∞
11	29.75228804	-82.26321990	29.75260181	-82.26277899	43.5

Table 5-3: The time based step response metrics recorded between segments 6 and 7 in all of the stable test runs. The values vary between controllers, but remain comparable.

Run Label	Response Time (sec)	Percent Overshoot	Settling Time (sec)	Steady State Error (m)
GPSD Run 1	4.3	38.4	20.8	-0.34
GPSD Run 2	4.4	39.7	20.4	-0.28
RD Run 1	8.6	8.3	20.6	-0.67
RD Run 2	8.8	0	19.7	-0.79

Figure 5-1: An aerial photograph of the Gainesville Raceway road course. The track has a variety of geometry that allows for accurate and controlled testing of an autonomous ground vehicle.

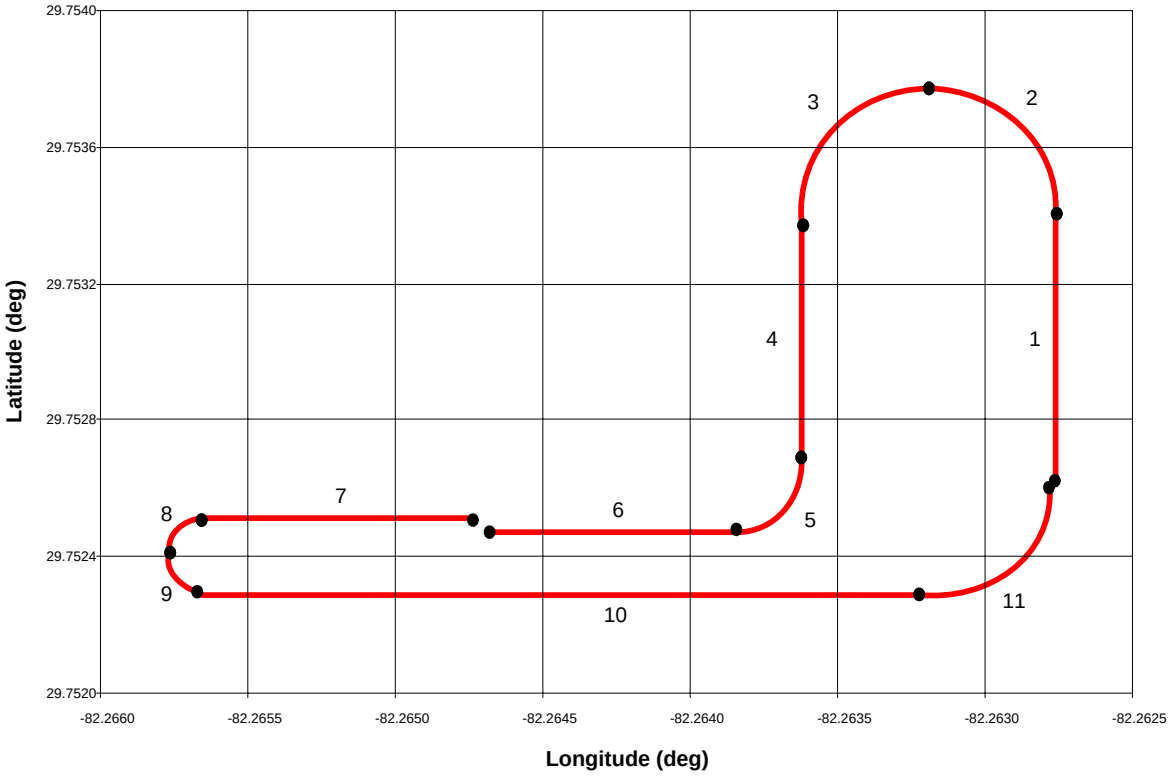

Figure 5-2: The path segments designed for the testing conducted at the Gainesville Raceway road course. A total of 11 segments make up the course, each with varying curvature. They are plotted over a geo-referenced global grid.

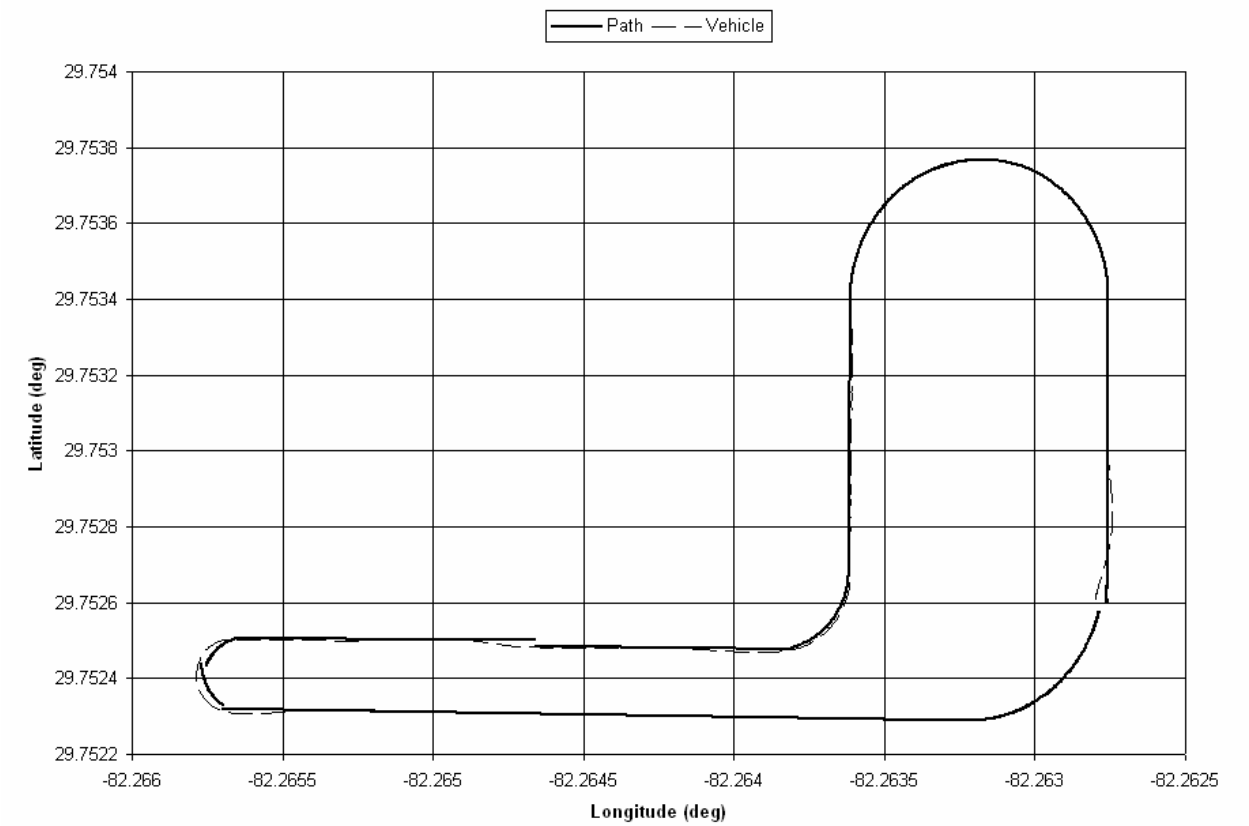

Figure 5-3: The position data collected from run 1 of test part 1. Both the vehicle's position and the planned path geometry are plotted over a geo-referenced grid.

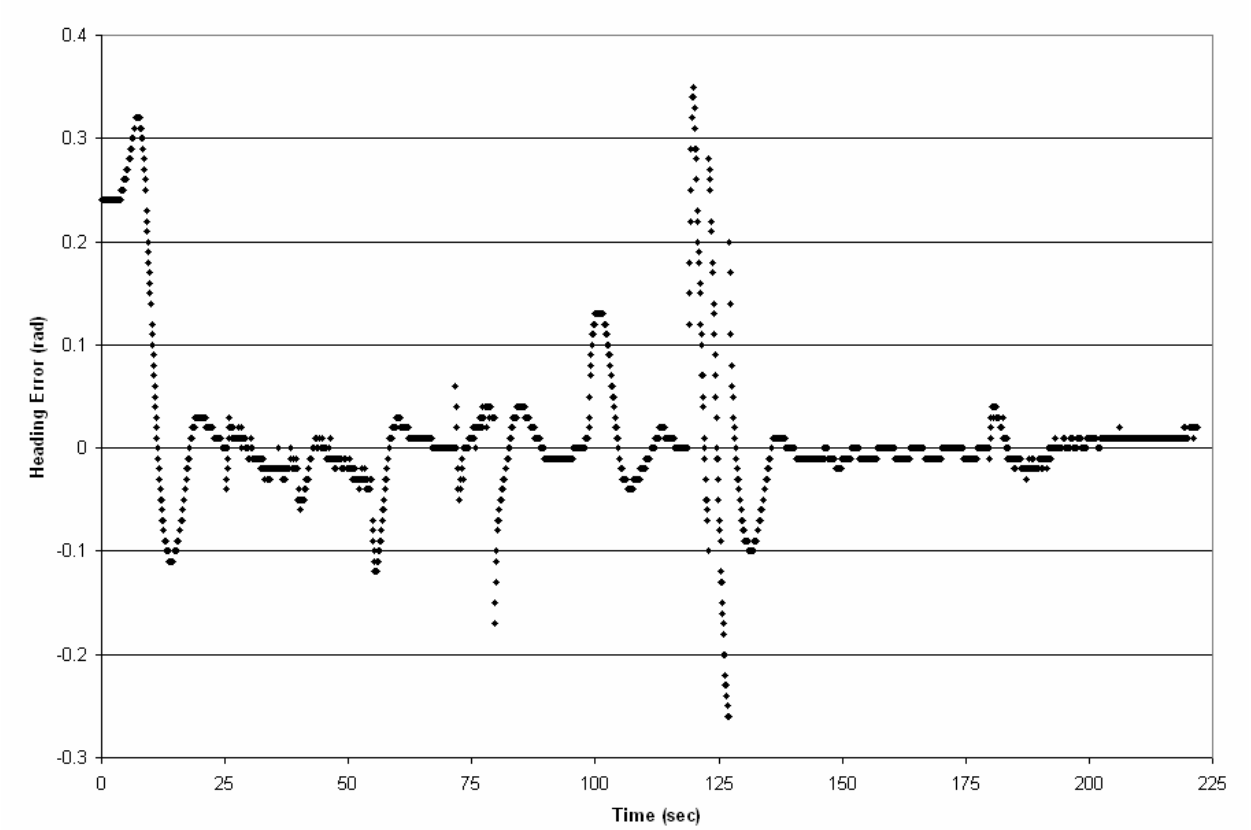

Figure 5-4: The logged NaviGator heading signal from test part 1 run 1. The signal starts off with an initial error value around 0.25 radians, and then regulates close to zero. There are disturbances during the 6 to 7 segment transition and at the sharp turn portion of the track.

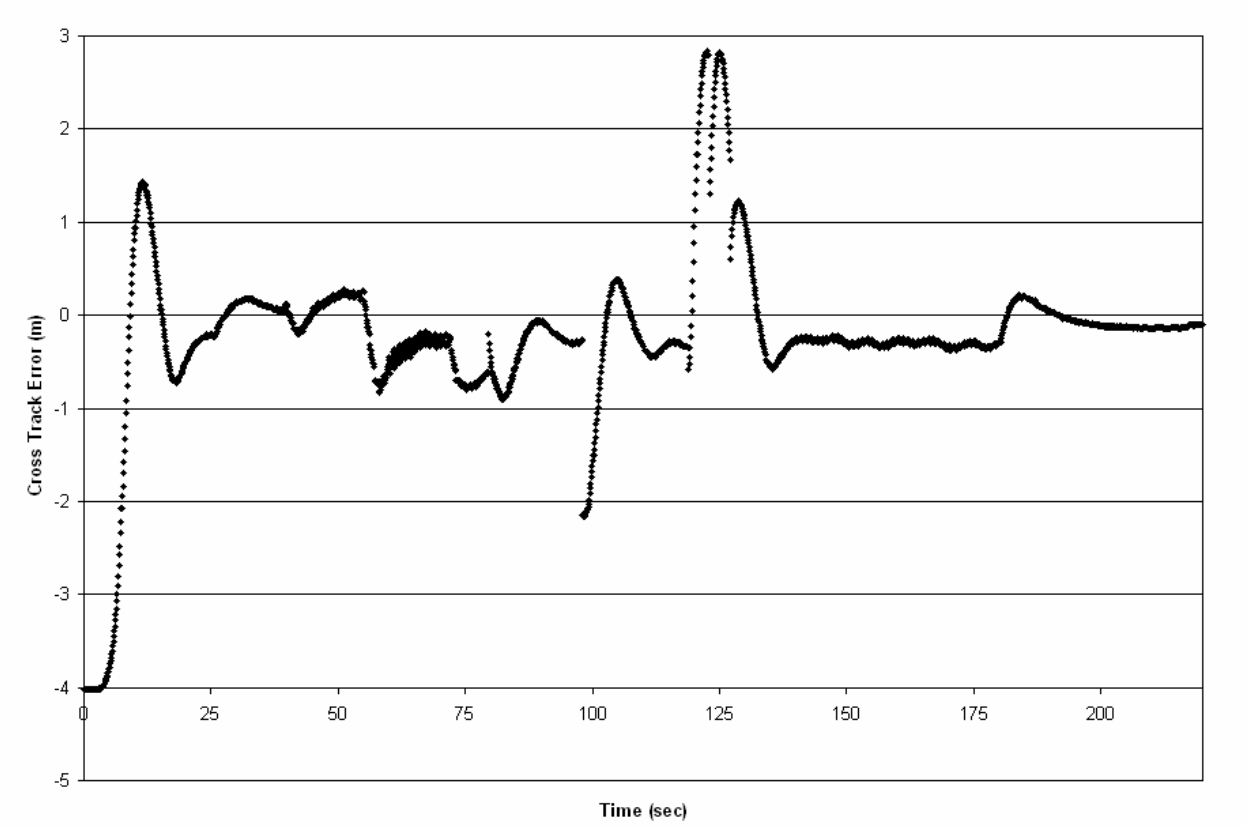

Figure 5-5: The cross track error signal from run 1 test part 1. The cross track error is initially around 4 meters and then converges toward zero. A disturbance is introduces at approximately 100 seconds, and the sharp turn creates a few large peaks shortly there after. The system is shown to track well here with steady state errors averaging less than 0.5 meters.

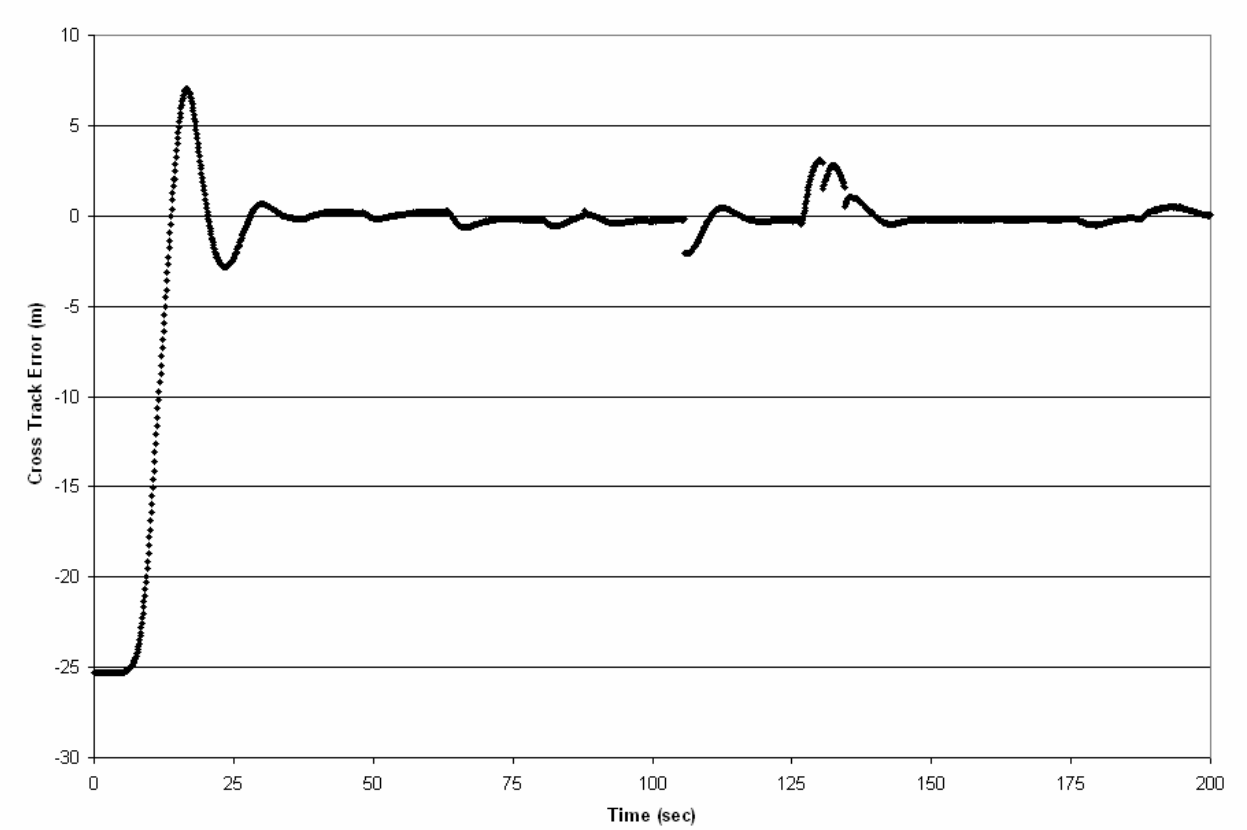

Figure 5-6: The cross track error signal from test part 1 run 2. There is purposely a large initial error, about 25 m, which is quickly regulated away by the PD controller, and the system remains stable. The remaining parts of the mission are executed in a fashion very similar to run 1, which is indicated by the timing and magnitude of the signal shown here.

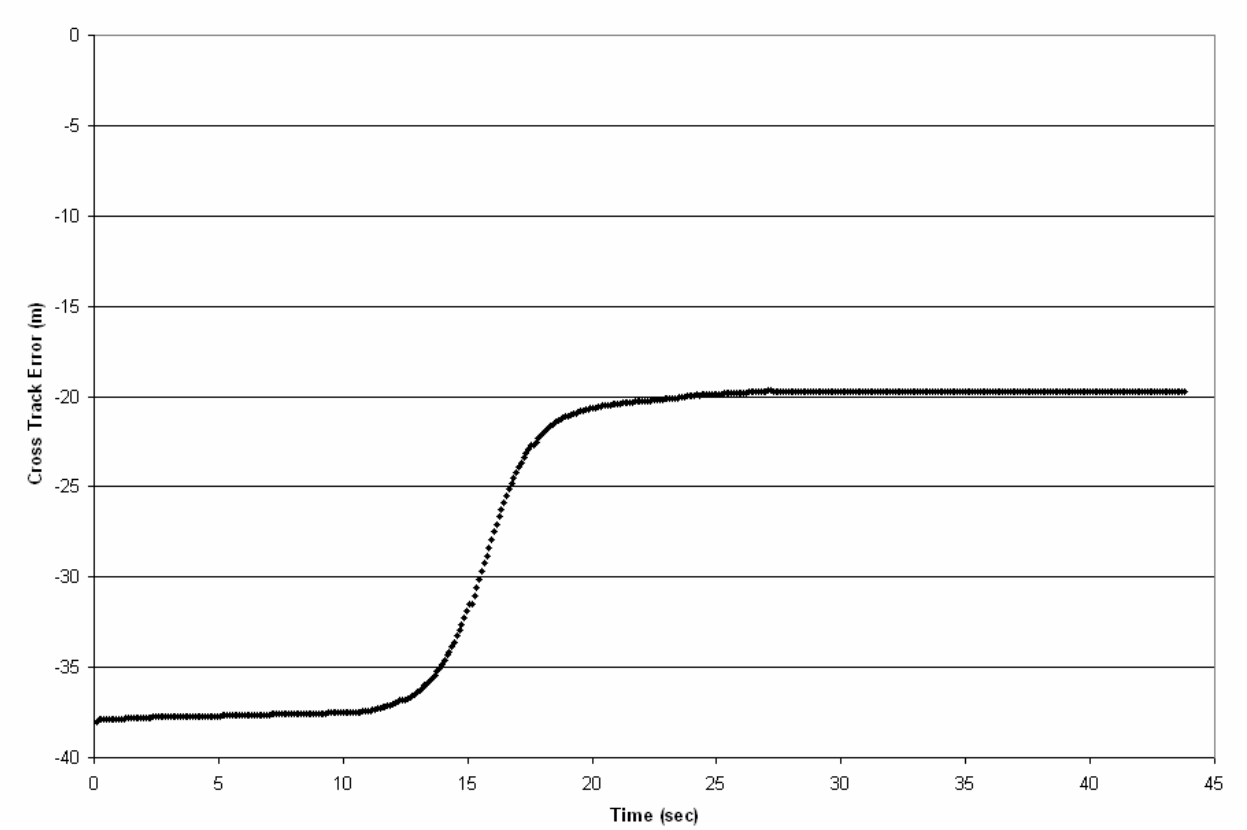

Figure 5-7: The cross track error signal from test part 1 run 3. The initial signal is two large for the PD controller to regulate and therefore causes the system to stay at -20 meters. The run was only conducted for approximately 30 seconds and then terminated for safety concerns.

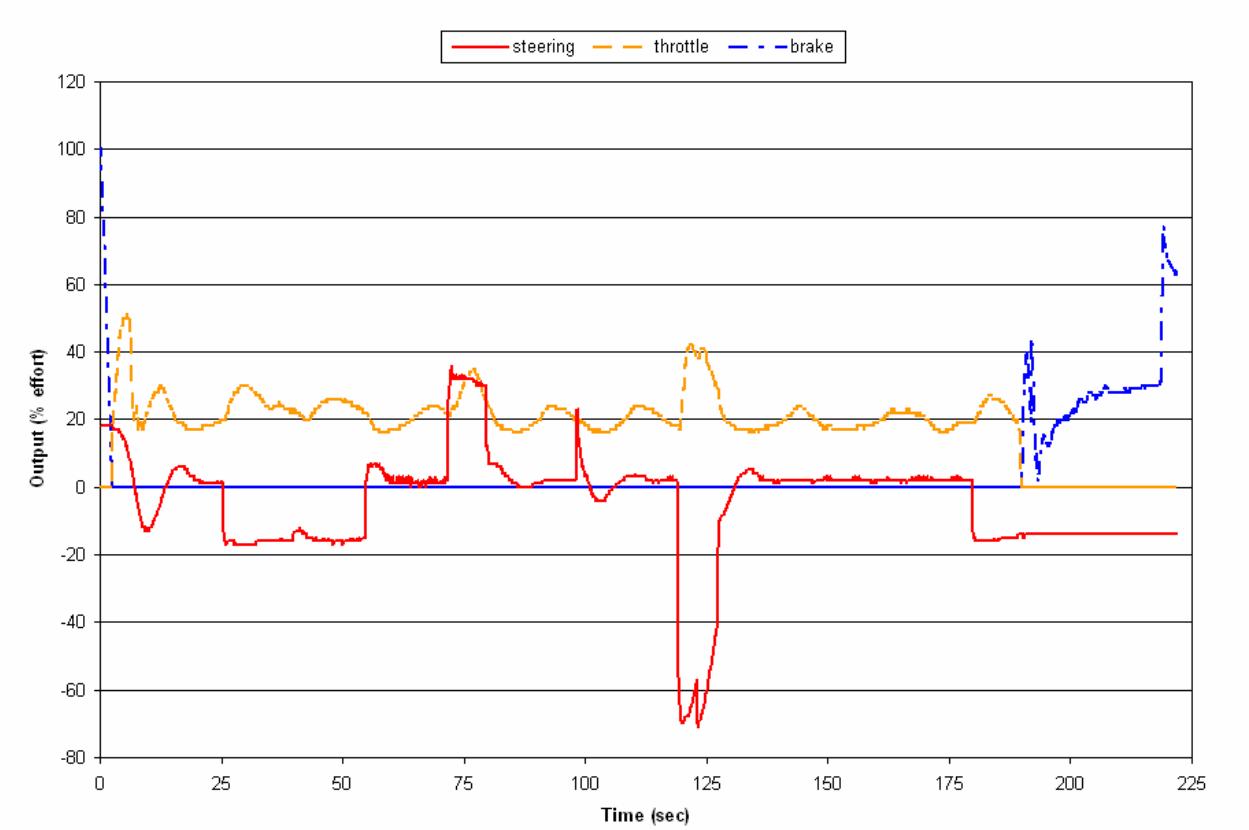

Figure 5-8: The output of all three wrench effort signals: steering throttle and brake, during run 1, test part 1. The steering signal is mostly smooth, with sharp discontinuities at segment transitions where curvature feed-forward dominates. The speed control signals are given for reference and demonstrate the speed system's limit cycle like behavior.

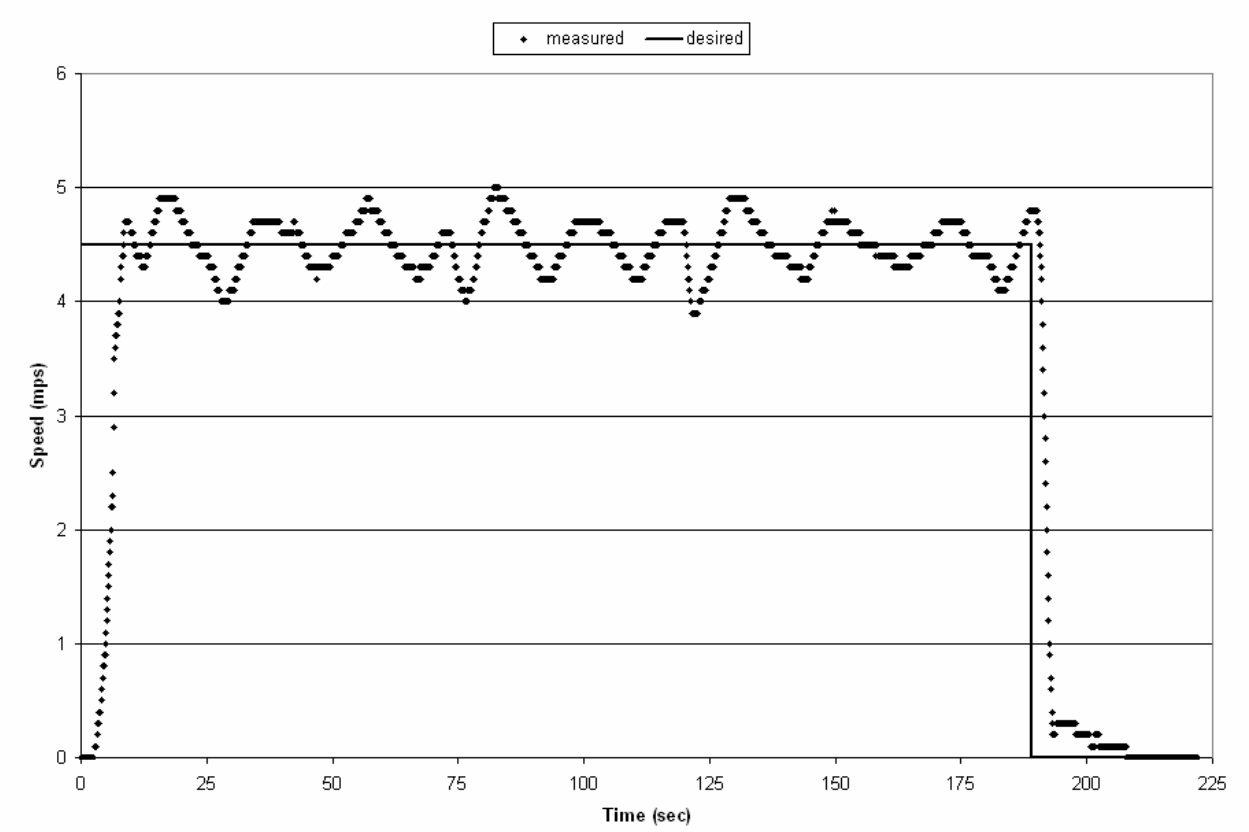

Figure 5-9: The speed controller performance data logged from run 1 test part 1. Notice that the speed measurement reaches an oscillating but stable limit cycle, even though the command remains constant. This is attributable to power train system nonlinearities and complex dynamics.

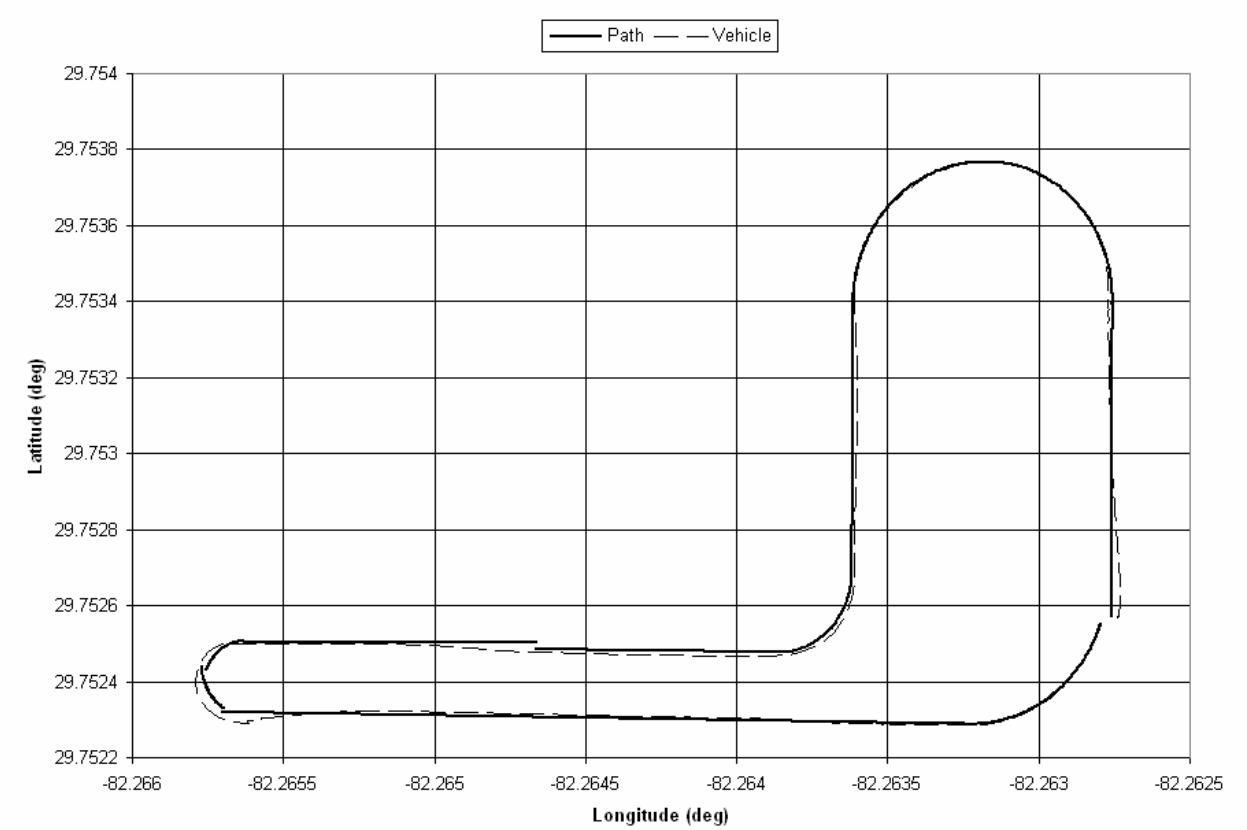

Figure 5-10: The position data collected from run 1 of test part 2. Both the vehicle's position and the planned path geometry are plotted over a geo-referenced grid. This run's performance data is very similar to the previous controller, with the exception of a larger steady state cross track error.

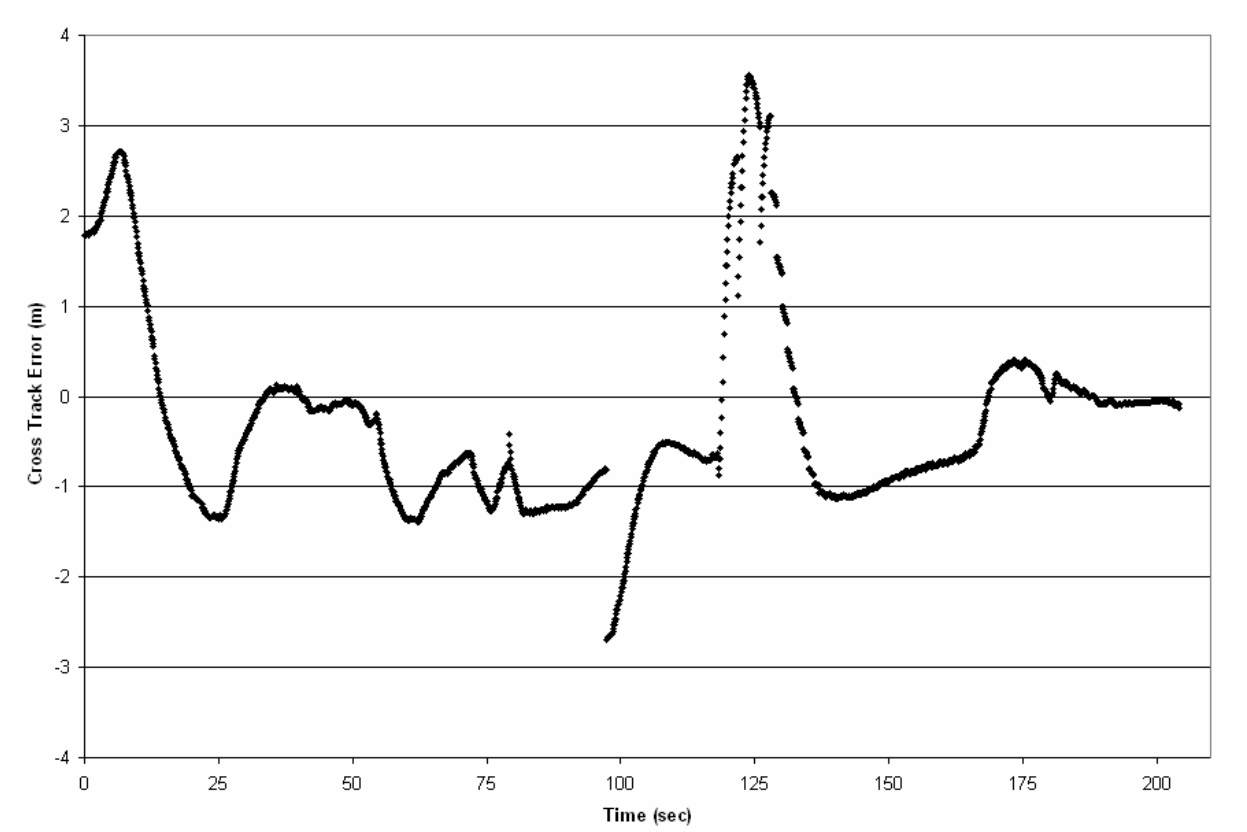

Figure 5-11: The cross track error signal from test part 2, run 1. Similar to the previous cross track signals, shown here is the system's performance while using the RD receding horizon controller. There is a discontinuity around 100 seconds, due to path step input, and also a large error peak due to the last sharp turn at 125 seconds.

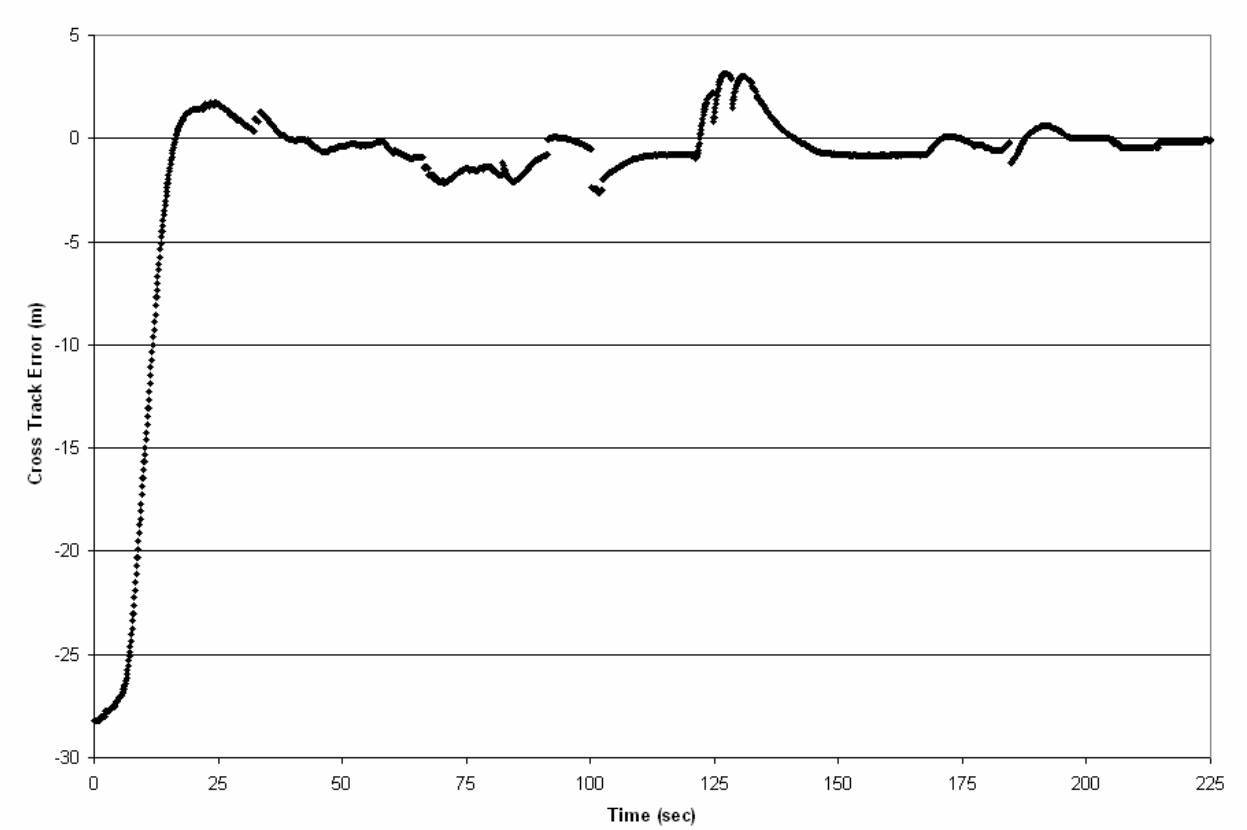

Figure 5-12: The cross track error log from run 2, test part 2. An intended large initial error is given, about 28 m. The RHC system regulates this directly down to near zero, with minimal overshoot. From this point on the controller performs almost identically to the previous run.

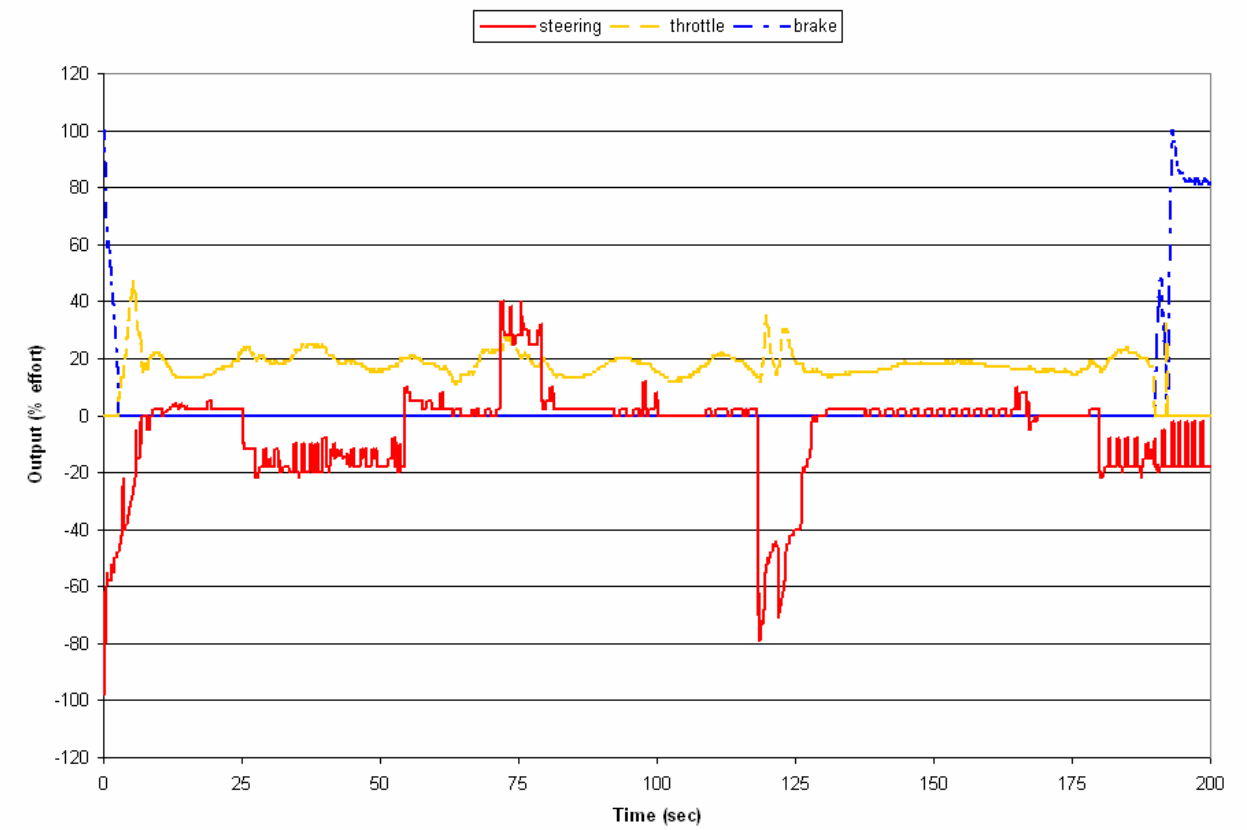

Figure 5-13: Each output signal logged during test part 2, run 1 (the nominal case). The steering control signal is much different than the one seen in the classic controller. It is much larger in certain cases and much smaller in others. One key feature is the input quantization, which is implied by the square wave like signals during the wide turns and long straight-aways.

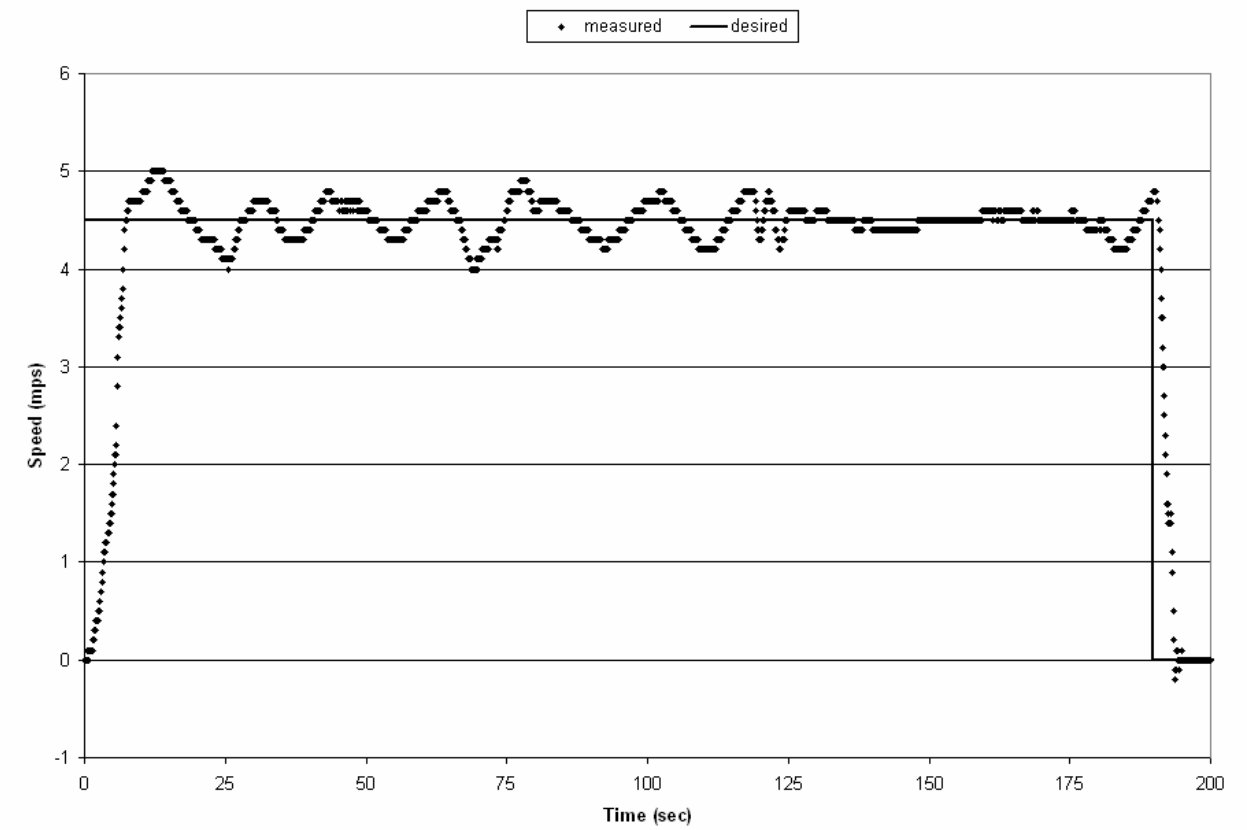

Figure 5-14: The speed control system data logged during run 1 of test part 2. Again the system demonstrates a stable limit cycle, as it did in the previous demonstration. This time the signal relaxes for a period, the cause of which is unknown. It is probably attributable to some nonlinearity in the system dynamics.

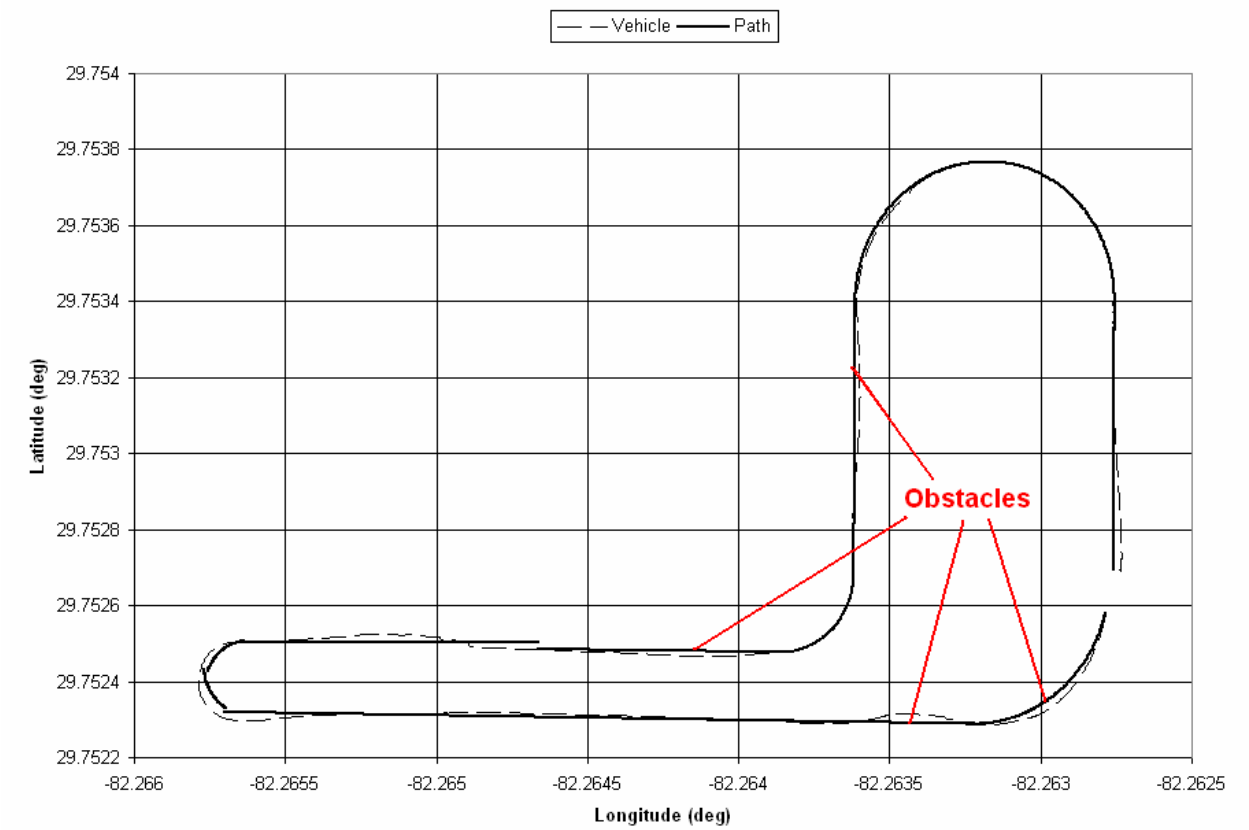

Figure 5-15: The position data collected from run 1 of test part 3. The obstacle locations during this run are called out with red leaders. Notice that the vehicle position deviates from the path around these areas.

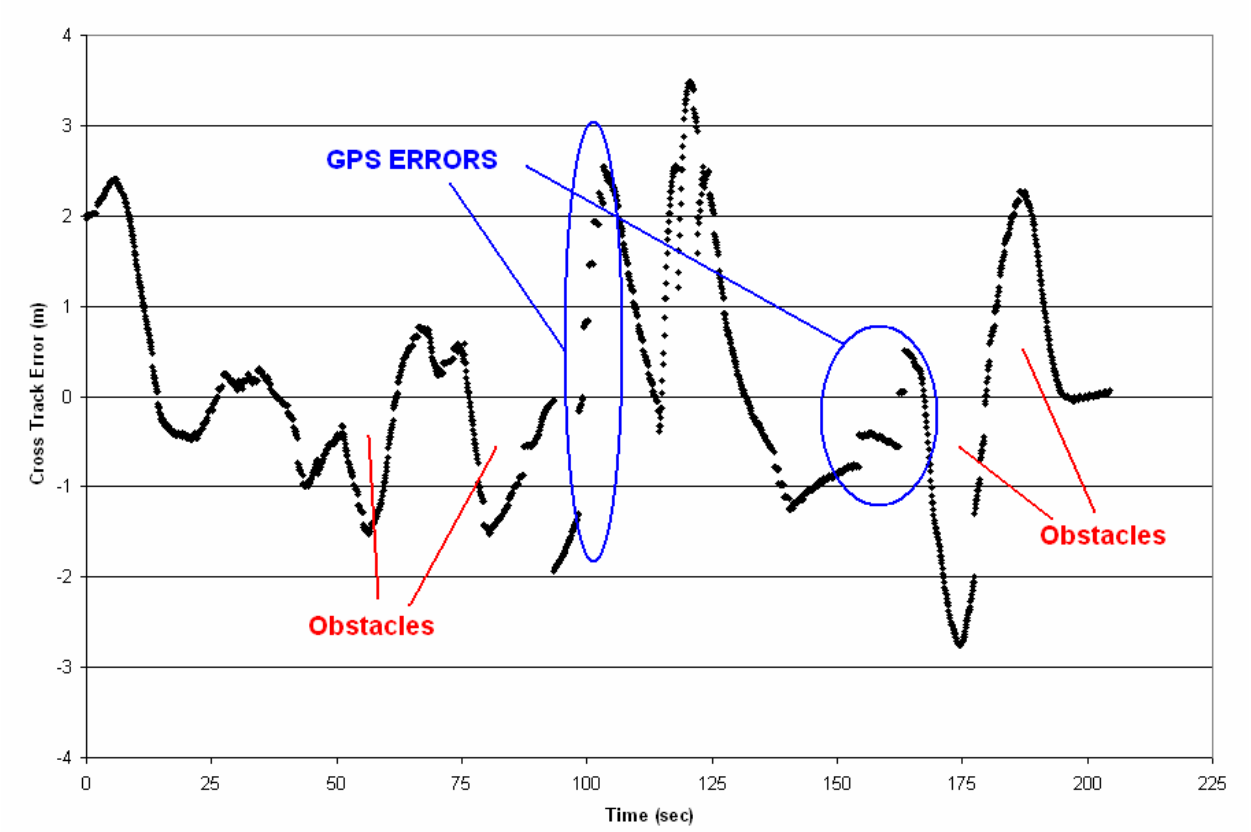

Figure 5-16: The cross track error log from run 1, test part 3. The obstacle locations during this run are called out with red leaders. Notice that the error signal increases and then decreases at these points. Also a group of GPS position solution spikes are called out and circled in blue.

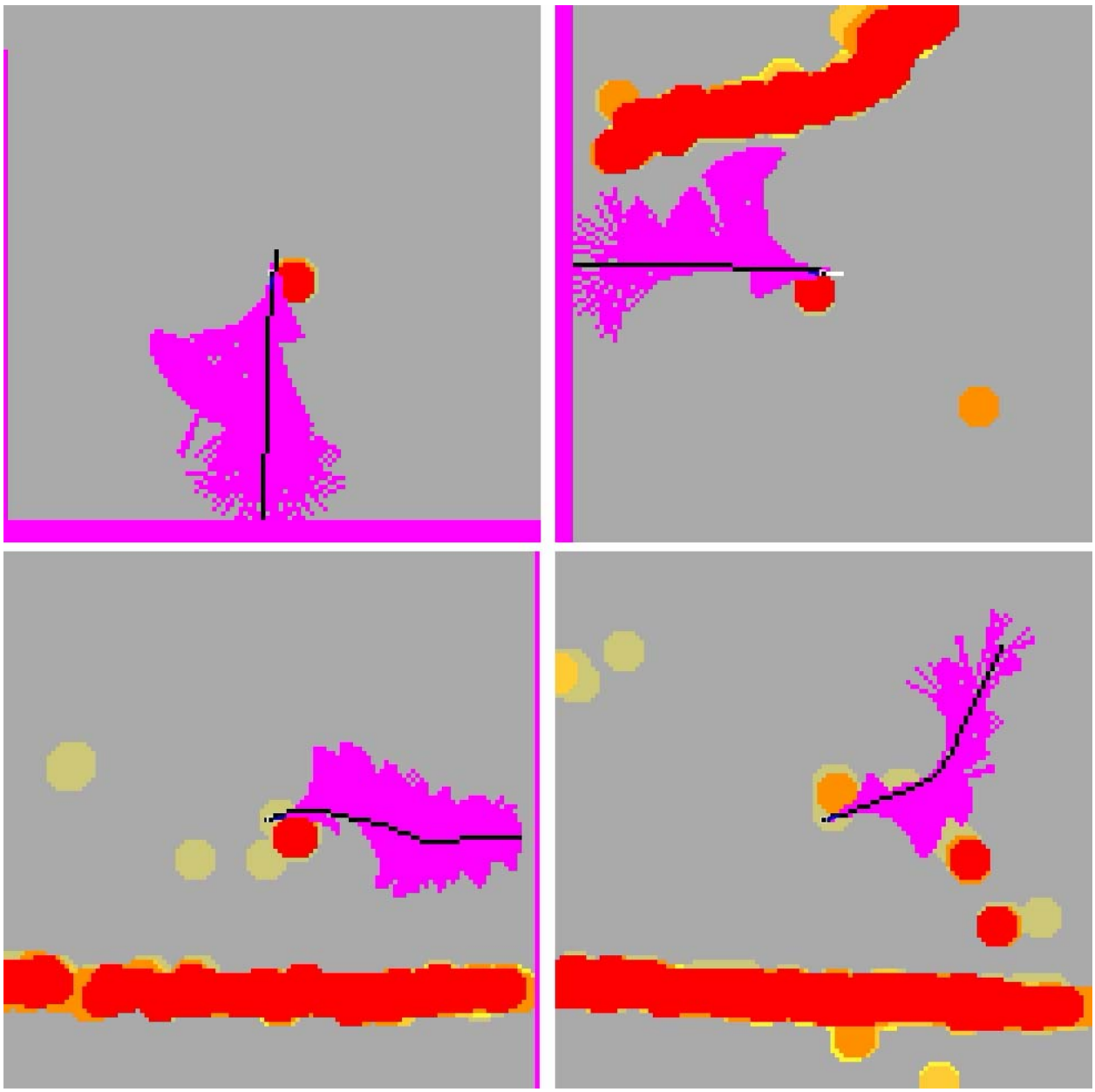

Figure 5-17: Four traversability grids recorded during run 1 of test part 3. Barrel 1 is avoided first (top left), then the 2nd (top right), the 3rd (bottom left), and finally the last barrel (bottom right). Free space is indicated in gray, with obstacles in shades of red and yellow. The pink region is space searched by the RHC algorithm, while the pink edges represent unknown space.

CHAPTER 6
FUTURE WORK AND CONCLUSIONS

This dissertation has introduced a new and novel planning and control scheme for autonomous systems. The method, called Heuristic Receding Horizon Control (HRHC) is used to simultaneously plan and control an autonomous system through its operating environment. Also a novel extension to classic Receding Horizon Control (RHC), called Dual-Frequency Receding Horizon Control (DFRHC) has been proposed.

The focus and motivation for the two technologies has been established in Chapter I, along with an initial problem statement. Specifically, it is desired to establish a method by which an autonomous ground vehicle may control its own motion through a cluttered environment. Chapter II has reviewed much of the relevant literature published by other researchers in this field, and has thus helped to establish the technology's uniqueness and niche within the realm of robotics and automation. The theory behind the HRHC and DFRHC concepts was offered and detailed in the third Chapter. In addition, the key proposal of this dissertation was summarized in Remark 1 of that chapter. Also both of the new and novel methods were implemented and tested on a real autonomous ground vehicle, the NaviGator, at the Center for Intelligent Machines and Robotics. The tested performance results for that implementation were given in Chapter IV, and were found to strongly support the theory and claims proposed.

Some advanced ideas and concepts, along with some remaining work that can be done to continue this research in the future are offered in the first section below. They are followed in the second section with some of the detailed conclusions that can be drawn from studying the theory, implementation and testing results of this dissertation.

Future Work

There are several concepts and advancements that can be made and tested for the RHC strategies of this dissertation. One comes from an implied hypothesis that both HRHC and DFRHC technique allow for RHC to be applied faster and more efficiently than a classic RHC optimization routine. While these techniques have been applied, tested, and shown to work on an electromechanical system, they have not been compared directly with a classic optimizer.

For this hypothesis, the modified RHC routine would ideally be tested and controlled against one that uses traditional optimization methods: Branch and Bound, Quadratic Programming, etc. Since these algorithms are very time consuming and complicated to implement, performing a thorough analysis of the new RHC algorithm against a classic one has been determined to be beyond the scope of this dissertation.

The testing would require measuring the computation performance of the algorithms themselves. Metrics that directly support the claim such as control loop frequency, bandwidth, and data throughput, can offer valuable input with regard to the algorithm's efficiency and effectiveness. For example, since HRHC requires the expansion of state representing nodes, the number of nodes searched in the process is another very good metric to evaluate the algorithm's efficiency, especially since the time performance and computation requirements are directly proportional to the number of nodes expanded. This type of testing would be very valuable to support the work begun in this thesis and offers a specific path for future research.

Another conceptual area could be focused on finding more advanced and theoretically supported ways of input quantization in order to increase the solution optimality of the HRHC routine. As discussed in the previous chapters, input quantization is required for the A* search, but diminishes the quality of the overall control. Since the quantization is currently generated *ad hoc*; further research in this area could certainly make the control algorithm and theory stronger.

Finally, many optimization routines implemented for RHC are able to use the previously determined solution trajectory as a warm or running start for the next optimization attempt. Since the state describing variables tend to change only slightly between control loop iterations, the previous solution makes for a very good initial guess. The HRHC routine implemented in this thesis does not use this technique, mostly because the solutions were shown to provide satisfactory control, and the algorithm is able to maintain real-time performance. However, research into giving the HRHC routine a warm start, could offer increased efficiency, which would allow for more nodes to be searched and a more optimal solution to be found. This result could then let the algorithm be applied to even more complex systems.

Conclusions

Motion planning and controlling an autonomous ground vehicle are very challenging tasks. These tasks are difficult even when accomplished in two sequential steps, which is the conventional practice in design and implementation. This dissertation has studied, theorized, implemented, and tested the unification of the two tasks into one. It can be drawn from this work that their combination indeed offers a significant amount of engineering consolidation. This is especially true when faced with the often daunting system integration effort required to create a functional real world robotic system. All of the signals, interconnections, and validations that must be made between the otherwise two sequential processes are avoided. This single task therefore, allows for less time to be spent in the development phase of the implementation.

One drawback in general is that the technology still needs some theoretical and practical hardening. It is not yet a tried and true methodology, and it is not simple. Even after the development stage, tuning a Receding Horizon Controller is difficult and vague. The engineers working with it must have a thorough knowledge of the system dynamics and nuances, as well as an advanced understanding of the underlying optimization routine. However, because the RHC is

more informed of the system dynamics, stronger control system performance can be achieved with the technique. Both system stability and robustness can benefit from the nonlinear capabilities of the controller.

Finally, the theoretical concepts of this thesis have overall been in effort to help unify the two largely accepted approaches to advanced robotics, namely, the Artificial Intelligence (AI) approach and the Control Systems approach. The AI approach is mainly logical in nature, and is based around proving that theoretical methods will yield admissible results, or in other words results which are valid, predictable and consistent. The controls approach is largely mathematical in nature and is concerned with the system dynamics and proving that formulaic controllers will yield stable and convergent behavior. Clearly the two methods must be related somehow, since the end results tend to be very similar, and because of the strong connections that analytical and logical mathematics share. This dissertation has shown that for one particular control method, an AI approach, A* search, and a controls method RHC, can be unified if the problem is structured and cast in a particular way. This is an important result identifying that the two largely practiced approaches may be more related than they appear.

LIST OF REFERENCES

[AN04] An, D., Wang, H., "VPH: A New Laser Radar Based Obstacle Avoidance Method for Intelligent Mobile Robots," *Proceedings. 2004 IEEE World Congress on Intelligent Control and Automation*, Hangzhou, China, June (2004), pp. 4681-4685.

[AND83] Andrews, J.R., Hogan, N., "Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator," *Control of Manufacturing Processes and Robotic Systems*, Eds. Hardt, D.E. and Book, W., ASME, Boston, 1983, pp. 243-251.

[BAL00] Ball, Sir R.S., *A Treatise on the Theory of Screws*, Cambridge, UK, Cambridge University Press, 1900.

[BAR92] Barraquand, J., Langlois, B., Latombe, J.-C., "Numerical Potential Field Techniques for Robot Path Planning," *IEEE Transactions on Systems, Man and Cybernetics*, Volume 22, Issue 2, March-April (1992), pp. 224-241.

[BEM00] Bemporad, A., Mignone, D., "MIQP.M: A Matlab Function for Solving Mixed Integer Quadratic Programs," *Technical Report*, code available at http://control.ethz.ch/~hybrid/miqp, Zurich, September 2006.

[BOR89] Borenstein, J., Koren, Y., "Real-Time Obstacle Avoidance for Fast Mobile Robots," *IEEE Transactions on Systems, Man, and Cybernetics*, Volume 19, Issue 5, October (1989), pp. 1179-1187.

[BOR91] Borenstein, J., Koren, Y., "The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots," *IEEE Transactions on Robotics and Automation*, Volume 7, Issue 3, June (1991), pp. 278-288.

[BRE63] Bresenham, J.E. "An Incremental Algorithm for Digital Plotting," Presented at ACM National Conference, August (1963).

[BRO83] Brockett, R.W., "Asymptotic Stability and Feedback Stabilization," in *Differential Geometric Control Theory*, R. W. Brockett, R. S. Millman, H. J. Sussmann (Eds.), Birkhauser, Boston, MA, (1983), pp. 181-191.

[BRO00] Brockett, R., Liberzon, D., "Quantized Feedback Stabilization of Linear Systems," *IEEE Transactions on Automatic Control*, Volume 45, Issue 7, July (2000), pp. 1279-1289.

[COU92] Coulter, C., "Implementation of the Pure Pursuit Path Tracking Algorithm," Report CMU-RI-TR-92-01, Carnegie Mellon University, Pittsburg PA, 1992.

[CHO00] Choi, S.B., "The Design of a Look-Down Feedback Adaptive Controller for the Lateral Control of Front-Wheel-Steering Autonomous Highway Vehicles," *IEEE Transactions on Vehicular Technology*, Volume 49, Issue 6, November (2000), pp. 2257-2269.

[CRA06] Crane, C. D., Armstrong, D.G., Touchton, R., Galluzzo, T., Solanki, S., Lee, J., Kent, D., Ahmed, M., Montane, R., Ridgeway, S., Velat, S., Garcia, G., Griffis, M., Gray, S., Washburn, J., Routson, G., "Team CIMAR's NaviGATOR: An Unmanned Ground Vehicle for Application to the 2005 DARPA Grand Challenge." *Journal of Field Robotics*, Volume 29, Issue 9, September (2006), pp. 599-623.

[DEL90] Delchamps, D.F., "Stabilizing a Linear System with Quantized State Feedback," *IEEE Transactions on Automatic Control*, Volume 35, Issue 8, August (1990), pp. 916-924.

[DEL98] De Luca, A., Oriolo, G., Samson, C., "Feedback Control of a Nonholonomic Car-like Robot," Chapter 4 of: Laumond, J-P., *Robot Motion Planning and Control*. Lectures Notes in Control and Information Sciences 229: Springer-Verlag Telos, 1998.

[DIX00] Dixon, W.E., Dawson, D.M., Zergeroglu, E., Behal, A., *Nonlinear Control of Wheeled Mobile Robots*, Vol. 262 Lecture Notes in Control and Information Sciences, Springer-Verlag London, 2000.

[DIX05] Dixon, W.E., Galluzzo, T. Hu, G., Crane, C., "Adaptive Velocity Field Control of a Wheeled Mobile Robot." *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo '05*, Dymaczewo, Poland, June (2005), pp. 145-150.

[DUB57] Dubins, L.E., "On Curves of Minimal Length with a Constraint on Average Curvature, and the Prescribed Initial and Terminal Positions and Tangents." *American Journal of Mathematics*, (1957), 79(3): pp. 497-516.

[ELF86] Elfes, A., "A Sonar-based Mapping and Navigation System." *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, April (1986), pp. 1151-1156.

[ELF89] Elfes, A., "Using Occupancy Grids for Mobile Robot Perception and Navigation." *IEEE Computer*, Volume 22, Issue 6, April (1989), pp. 46-57.

[FIO93] Fiorini, P., Shiller, Z., "Motion Planning in Dynamic Environments Using the Relative Velocity Paradigm," *Proceedings. 1993 IEEE International Conference on Robotics and Automation*, May (1993), pp. 560-565.

[GU05] Gu, D., Hu, H., "A Stabilization Receding Horizon Regulator for Nonholonomic Mobile Robots," *IEEE Transactions on Robotics*, Volume 21, Issue 5, October (2005), pp. 1022-1028.

[HAR68] Hart, P.E., Nilsson, N.J., Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, SSC4 (2), (1968), pp. 100-107.

[ISI95] Isidori, A., *Nonlinear Control Systems*, 3rd Edition, London, UK: Springer-Verlag, 1995.

[JIA97] Jiang, Z-P., Nijmeijer, H., "Tracking Control of Mobile Robots: A Case Study in Backstepping," *Automatica*, Volume 33, Issue 7, (1997), pp. 1393-1399.

[JIN05] Jing, X-J., Wang, Y-C., "Control of Behavior Dynamics for Motion Planning of Mobile Robots in Uncertain Environments," *Proceedings. 2005 IEEE International Conference on Mechatronics*, Taipei, Taiwan, July (2005), pp. 364-369.

[KAM86] Kambhampati, S., Davis, L., "Multiresolution Path Planning for Mobile Robots," *IEEE Journal of Robotics and Automation*, Volume 2, Issue 3, September (1986), pp. 135-145.

[KAV96] Kavralu, L., Svestka, P., Latombe, J-C., Overmars, M., "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, Volume 12, Issue 4, August (1996), pp. 566-580.

[KIM01] Kim, C.S., Park, K.S., Jeong, S.G., Lee, W.G., Lee, H.C., Kim, J.C., Bae, J.I., Lee, M.H., "H ∞ Steering Control System for the Unmanned Vehicle," *Proceedings. 2001 IEEE International Symposium on Industrial Electronics*, Pusan, Korea, June (2001), pp. 1441-1445.

[KEY94] Keymeulen, D., Decuyper, J., "The Fluid Dynamics Applied to Mobile Robot Motion: The Stream Field Method," *Proceedings. 1994 IEEE International Conference on Robotics and Automation*, May (1994), pp. 378-385.

[KHA85] Khatib, O., "Real-time Obstacle Avoidance for Manipulators and Mobile Robots," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, Mar (1985), pp. 500-505.

[KON99] Konkimalla, P., LaValle, S.M., "Efficient Computation of Optimal Navigation Functions for Nonholonomic Planning." *Proceedings of the First Workshop on Robot Motion and Control, 1999. RoMoCo '99*, Kiekrz, Poland, June (1999), pp. 187-192.

[KOR91] Koren, Y., Borenstein, J., "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, April (1991), pp. 1398-1404.

[KUA85] Kaun, D., Zamiska, J., Brooks, R., "Natural Decomposition of Free Space for Path Planning," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, March (1985), pp. 168-173.

[LI93] Li, P., Horowitz, R., "On Velocity Field Control of Mechanical Systems," *Proceedings. 1993 IEEE Conference on Decision and Control*, December (1993), pp. 3124-3125.

[LI96] Li, P., Horowitz, R., "Application of Passive Velocity Field Control to Robot Contour Following Problems," *Proceedings. 1996 IEEE Conference on Decision and Control*, December (1996), pp. 378-385.

[LIN04] Lingelbach, F., "Path Planning using Probabilistic Cell Decomposition," *Proceedings. 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, April (2004), pp. 467-472.

[LIU94] Liu, Y-H., Arimoto, S., "Computation of the Tangent Graph of Polygonal Obstacles by Moving-Line Processing," *IEEE Transactions on Robotics and Automation*, Volume 10, Issue 6, December (1994), pp. 823-830.

[LOI03] Loizou, S.G., Tanner, H.G., Kumar, V., Kyriakopoulos, K.J., "Closed Loop Navigation for Mobile Agents in Dynamic Environments." *Proceedings. 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, NV, October (2003), pp. 3769-3774.

[LOZ79] Lozano-Perez, T, Wesley, M., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, Volume 22, Issue 10, October (1979), pp. 560-570.

[MAY90] Mayne, D.Q., Michalska, H., "Receding Horizon Control of Nonlinear Systems," *IEEE Transactions on Automatic Control*, Volume 35, Issue 7, July (1990), pp. 814-824.

[MAY00] Mayne, D. Q., Rawlings, J. B., Rao, C. V., Scokaert, P. O. M., "Constrained model predictive control: Stability and optimality," *Automatica*, Volume 36, Issue 6, June (2000), pp. 789-814.

[MEY86] Meystel, A., Guez, A., Hillel, G., "Minimum Time Path Planning for a Robot," *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, April (1986), pp. 1678-1687.

[MIU99] Miura, J., Uozumi, H., Shirai, Y., "Mobile Robot Motion Planning Considering the Motion Uncertainty of Moving Obstacles," *Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics*, Tokyo, Japan, October (1999), pp. 692-697.

[NEL88] Nelson, W.L., Cox, I.J., "Local Path Control for an Autonomous Vehicle," *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, April (1988), pp. 1504-1510.

[NIC98] Nicolao, G.D., Magni, L., Scattolini, R., "Stabilizing Receding-Horizon Control of Nonlinear Time-Varying Systems," *IEEE Transactions on Automatic Control*, Volume 43, Issue 7, July (1998), pp. 1030-1036.

[NIL71] Nilsson, Nils, J. *Problem Solving Methods in Artificial Intelligence*. New York: McGraw-Hill, 1971.

[NIL98] Nilsson, Nils, J. *Artificial Intelligence: A New Synthesis*. San Francisco: Morgan Kaufman Publishers, Inc., 1998.

[OLL95] Ollero, A., Heredia, G., "Stability Analysis of Mobile Robot Path Tracking," *Proceedings. 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (1995), pp. 461-466.

[REE91] Reeds J.A., Shepp, R.A., "Optimal Paths for a Car that Goes Both Forward and Backward," *Pacific Journal of Mathematics* (1991), 145(2): pp. 367-393.

[RIM88] Rimon, E., Koditschek, D.E., "Exact Robot Navigation using Cost Functions: The Case of Distinct Spherical Boundaries in E^n," *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, April (1988), pp. 1791-1796.

[RIM91] Rimon, E., "A Navigation Function for a Simple Rigid Body," *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, April (1991), pp. 546-551.

[ROH87] Rohnert, H., "Shortest Paths in the Plane with Convex Polygonal Obstacles," *Information Processing Letters*, Volume 23, 1987, pp. 71-76.

[SAM90] Samson, C., "Velocity and Torque Feedback Control of a Nonholonomic Cart," *Proceedings. International Workshop in Adaptive and Nonlinear Control: Issues in Robotics*, Grenoble, France, 1990.

[SAM95] Samson, C., "Control of Chained Systems Application to Path Following and Time-Varying Point-Stabilization of Mobile Robots," *IEEE Transactions on Automatic Control*, Volume 40, Issue 1, January (1995), pp. 64-77.

[SCO99] Scokaert, P.O.M, Mayne, D.Q., Rawlings, J.B., "Suboptimal Model Predictive Control (Feasibility Implies Stability)," *IEEE Transactions on Automatic Control*, Volume 44, Issue 3, March (1999), pp. 648-654.

[TAK89] Takahashi, O., Schilling, R.J., "Motion Planning in a Plane Using Generalized Voronoi Diagrams," *IEEE Transactions on Robotics and Automation*, Volume 5, Issue 2, April (1989), pp. 143-150.

[THO84] Thorpe, C., Matthies, L., "Path Relaxation: Path Planning for a Mobile Robot," *Proceedings. 1984 IEEE Conference OCEANS*, September (1984), pp. 576-581.

[THR05] Thrun, S., Burgard, W., Fox, D., *Probabilistic Robotics*, Cambridge, MA: The MIT Press, 2005.

[TIL90] Tilove, R.B., "Local Obstacle Avoidance for Mobile Robots Based on the Method of Artificial Potentials," *Proceedings. 1990 IEEE International Conference on Robotics and Automation*, May (1990), pp. 566-571.

[US00] United States. Floyd D. Spence National Defense Authorization Act for Fiscal Year 2001. H.R. 5408. H.Rept. 106-945. Congressional Record. Washington: GPO, October 6, 2000.

[WAR89] Warren, C.W., "Global Path Planning using Artificial Potential Fields," *Proceedings. 1989 IEEE International Conference on Robotics and Automation*, Volume 1, May (1989), pp. 316-321.

[WIT00] Wit, J., *Vector Pursuit Path Tracking for Autonomous Ground Vehicles*. Ph.D. Dissertation, University of Florida, 2000.

BIOGRAPHICAL SKETCH

Thomas Galluzzo was born and raised in western New York. He holds a B.S. in aerospace engineering from Embry-Riddle Aeronautical University at Daytona Beach, FL. He is currently working on completing a doctoral degree from the University of Florida, focused on robotic ground vehicles. He plans to continue his career as an engineer at Harris Corporation in Melbourne, FL, after his graduate education.